Internet Draft
draft-dube-modbus-applproto-00.txt
Category: Informational

D. Dube
J. Camerini
Schneider Automation Inc.

May 2002

MODBUS Application Protocol

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on November 10, 2002.

Copyright Notice

Abstract

MODBUS is an application layer messaging protocol, positioned at level 7 **of the OSI model. MODBUS provides client/server communication between** devices connected on different types of buses or networks. MODBUS is a confirmed service protocol and offers many services specified by function codes. MODBUS function codes are elements of MODBUS Request/Reply PDUs. The objective of this document is to describe the function codes used within the framework of MODBUS/TCP transactions.

Comments
========

Please send comments to Dennis Dube at dennis.dube@modicon.com

Table of Contents
=================

1. Overview

MODBUS is an application layer messaging protocol for client/server communication. MODBUS exists on different communication stacks. This RFC describes MODBUS/TCP.

MB/TCP Protocol Stack

LAYER	PROTOCOL
Application	MODBUS
Transport	TCP
Network	IP
Link	Ethernet II & IEEE 802.3
Physical	10BaseT & 100BaseT

1.1 Abbreviations

ADU	Application Data Unit
HMI	Human Machine Interface
IETF	Internet Engineering Task Force
I/O	Input/Output
IP	Internet Protocol
MAC	Medium Access Control
MB	MODBUS Protocol
MBAP	MODBUS Application Protocol
PDU	Protocol Data Unit
PLC	Programmable Logic Controller
TCP	Transport Control Protocol

2.0 MODBUS/TCP Protocol Specification

This section presents the elements of the MODBUS/TCP Protocol which include :

* MODBUS/TCP PDU Structure
* MODBUS Data Encoding
* MODBUS Data Model
* MODBUS Transactions

The focus and scope of the entire section is the application layer.

2.1 MODBUS/TCP PDU Structure

The MODBUS protocol defines a simple protocol data unit (PDU)independent of the underlying communication layers. The mapping of MODBUS protocol on specific buses or networks can introduce some additional framing fields. The application data unit (ADU) for MODBUS is defined as the MODBUS PDU plus any framing fields. The ADU for MODBUS is defined as :

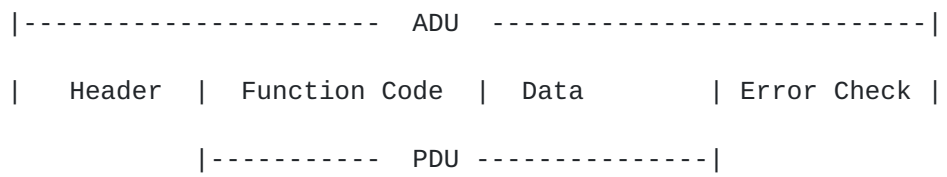


Figure 1 MODBUS ADU

The MODBUS/TCP ADU is defined as :

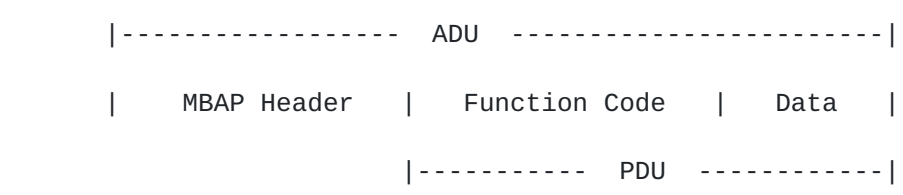
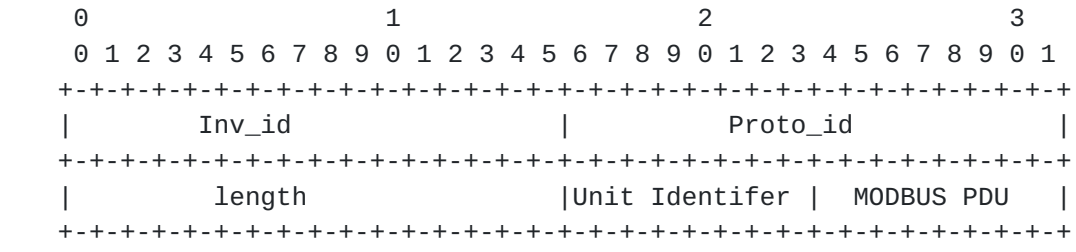


Figure 2 MODBUS/TCP ADU

A MODBUS/TCP ADU includes an MBAP header and a MODBUS PDU, and is defined as:



The MBAP header is 7 bytes long, and includes the following fields :

inv_id - [2 bytes] invocation identification used for transaction pairing

proto_id - [2 bytes] is always 0 for MODBUS services. Other values are reserved for further extensions.

length - [2 bytes] The length field is a byte count of the remaining fields and includes the dst_id and data fields.

unit_id - [1 byte] The unit identifier is used to identify a remote server located on a non-TCP/IP network.

All MODBUS/TCP ADUs are sent via TCP on registered system port 502.

The service portion of the MODBUS Application Protocol is the MODBUS PDU that contains 2 fields :

func_code - [1 byte] MODBUS function code

data - [n bytes] This field is function code dependent and usually contains information such as variable references,

variable counts, data offsets, sub-function codes etc.

The size of the MODBUS/TCP ADU is limited by the size constraint inherited from the first MODBUS implementations on Serial Line networks, eg the max. MODBUS/RS485 ADU = 256 bytes. Therefore, the MODBUS/TCP ADU is 256 and consequently the MODBUS/TCP PDU = 256 û 7 (length of MBAP Header) = 249 bytes max.

The MODBUS protocol defines three PDUs. They are :

* MODBUS Request PDU, mb_req_pdu
* MODBUS Response PDU, mb_rsp_pdu
* MODBUS Exception Response PDU, mb_excep_rsp_pdu

The mb_req_pdu is defined :

mb_req_pdu = { function_code, request_data), where

function_code - [1 byte] MODBUS function code

request_data - [n bytes] This field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function codes etc.

The mb_rsp_pdu is defined :

mb_rsp_pdu = { function_code, response_data), where

function_code - [1 byte] MODBUS function code

response_data - [n bytes] This field is function code dependent and usually contains information such as variable references, variable counts, data offsets, sub-function codes, etc.

The mb_excep_rsp_pdu is defined :

mb_excep_rsp_pdu = { function_code, request_data), where

function_code - [1 byte] MODBUS function code + 0x80

exception_code - [1 byte] MODBUS Exception Code Defined in table below.

Code	Name	Meaning
01	**ILLEGAL FUNCTION**	The function code received in the request

		is not an allowable action for the server.
02	**ILLEGAL DATA ADDRESS**	The data address received in the request is not an allowable address for the server.
03	**ILLEGAL DATA VALUE**	A value contained in the request data field is not an allowable value for server.
04	**SERVER FAILURE**	An unrecoverable error occurred when the server attempted to perform the requested action.
05	**ACKNOWLEDGE**	The server has accepted the request and is processing it. However a long duration of time will be required to finish. This response is returned to prevent a timeout error from occurring at the client. The client must use some polling technique to determine when the processing of the request is completed.
06	**SERVER BUSY**	The server is engaged in processing a longûduration program command. The client should retransmit the message later when the server is free.
08	**MEMORY PARITY ERROR**	Specialized use in conjunction with function codes 20 and 21 and reference type 6. Indicates that the extended file area failed to pass a consistency check. The server attempted to read extended memory, but detected a parity error in the memory. The client can retry the request, but service may be required on the server device.
0A	GATEWAY PATH UNAVAILABLE	Specialized use in conjunction with gateways. Indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
0B	GATEWAY TARGET DEVICE FAILED TO RESPOND	Specialized use in conjunction with gateways. Indicates that no response was obtained from the target device. Usually means that the device is not present on the network.
255	**EXTENDED EXCEPTION**	Indicates the mb_exception_rsp_pdu contains extended exception information. A subsequent

RESPONSE	two-byte length field indicates the size in bytes of the function code specific exception information.

2.2 MODBUS Data Encoding

MODBUS uses a æbig-EndianÆ representation for addresses and data items. This means that when a numerical quantity larger than a single byte is transmitted, the most significant byte is sent first. So for example

Register size	Value	
16 - bits	0x1234	the first byte sent is 0x12 then 0x34

Some function codes can use other encoding rules, However in this case the encoding rule has to be described explicitly.

Please see RFC 791, Appendix B, for a detailed description of the default data encoding and transmission order.

2.3 MODBUS Data Model

MODBUS bases its data model on a series of indexed register files that have distinguishing characteristics. The four primary register file types are:

Register Type	Data Type	Access	Comments
Discrete Input	single bit	read-only	data from I/O system
Discrete Output(Coils)	single bit	read-write	data from application
Input Registers	16-bit word	read-only	data from I/O system
Holding Registers	16-bit word	read-write	data from application

The distinctions between inputs and outputs, and between bit-addressable and word-addressable data items, do not imply any application behavior. It is perfectly acceptable, and very common, to regard all four register files as overlaying one another, if this is the most natural interpretation on the target machine in question.

For each of the primary register files, the protocol allows individual selection of 65536 data items, and the operations of read or write of those items are designed to span multiple consecutive data items up to a data size limit which is dependent on the transaction function code.

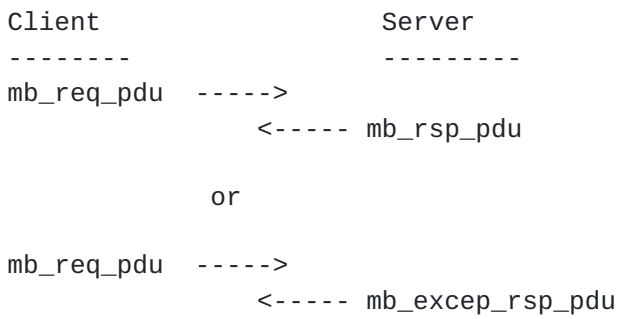
Data handled via MODBUS (bits, registers, etc) is usually located in device application memory, however physical addresses in memory should not be confused with MODBUS data references. A device is only required

to link MODBUS data references with physical addresses.

MODBUS logical reference numbers, which are used in MODBUS functions, are unsigned integer indices starting at zero.

2.4 MODBUS/TCP Transactions

MODBUS is a confirmed service protocol and consequently MODBUS/TCP transactions are request/response pairs of PDUs. Two scenarios are possible, ie after sending a mb_req_pdu to the MODBUS server, the MODBUS client will receive either a normal mb_rsp_pdu or a mb_excep_rsp_pdu in return .



The following pseudo code describes the processing of a mb_req_pdu received by a MODBUS server.

```
wait      wait until a mb_req_pdu is received

parse     Check mb_req_pdu
          if function code is invalid then go to excep_01
          if data address is invalid then go to excep_02
          if data values are invalid then go to excep_03

run       Execute function
          if function code does not execute OK then go to
                                                  excep_run
          send mb_rsp_pdu
          go to wait

excep_01  send mb_excep_rsp_pdu with exception code 1
          go to wait

excep_02  send mb_excep_rsp_pdu with exception code 2
          go to wait

excep_03  send mb_excep_rsp_pdu with exception code 3
```

```
                go to wait

        excep_run send mb_excep_rsp_pdu with an execution exception code
                  go to wait
```

For an exception response, the server returns a function code that is the original function code plus 0x80, ie with its most significant bit set to 1.

Client transaction timeout handling and mb_req_pdu retry mechanisms are not inherently part of the MODBUS protocol although many client applications implement such mechanisms.

3. Function Code Categories

There are 3 categories of MODBUS Function Codes, ie

1. Public Function Codes
2. User-Defined Function Codes
3. Reserved Function Codes

The set of Public Function Codes is actually a union of 2 sets of function codes. They are :

1. Assigned Function Codes, described in this RFC
2. Unassigned Function Codes, set aside for future assignment.

Public Function Codes

* are well defined function codes ,
* guaranteed to be unique,
* validated by the modbus.org community,
* publicly documented at modbus.org,
* have available conformance tests,
* are publicly documented by means of this IETF RFC for the MODBUS Application Protocol,
* include both assigned function codes as well as unassigned function codes reserved for future use.

User-Defined Function Codes

* There are two defined ranges of user-defined function codes, ie
* 65 to 72 decimal (0x41 to 0x58), and
* 100 to 110 decimal (0x64 to 0x6E).
* The user can select and implement a function code without any approval from modbus.org.
* There is no guarantee that the use of the selected function code will be unique.

Reserved Function Codes

* are currently in use in various products,
* are not considered part of the set of Public Function Codes,
* are not assignable as Public Function Codes.

3.1 Assigned Public Function Codes

Data Access

Bit Access

Function Name	Function Code
Read Discrete Inputs	02 (0x02)
Read Coil Status	01 (0x01)
Force Single Coil	05 (0x05)
Force Multiple Coils	15(0x0F)

16-bit Word Access

Function Name	Function Code
Read Input Registers	04 (0x04)
Read Holding Registers	03(0x03)
Write Single Register	06(0x06)
Write Multiple Registers	16(0x11)
Read/Write Holding Registers	23(0x17)
Mask Write Holding Register	22(0x16)

File Record Access

Function Name	Function Code	Sub-Function Code
Read File Record	20(0x14)	06(0x06)
Write File Record	21(0x15)	06(0x06)

Encapsulated Interface

Function Name	Function Code	MEI Type
Read Device Identification	43(0x2B)	14(0x0E)

4. Public Function Code Definitions

4.1 Read Coil Status 01 (0x01)

This function code is used to read from 1 to 2000 contiguous status of coils in a remote device. The Request PDU specifies the starting address, ie the address of the first coil specified, and the number of coils. Coils are addressed starting at zero. Therefore coils 1-16 are addressed as 0-15.

The coils in the response message are packed as one coil per bit of the data field. Status is indicated as 1= ON and 0= OFF. The LSB of the first data byte contains the output addressed in the query. The other coils follow toward the high order end of this byte, and from low order to high order in subsequent bytes.

If the returned output quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Request PDU

Function Code	1 byte	01 (0x01)
Starting Address	2 bytes	0x0000 to 0xFFFF
No. of Coils	2 bytes	1 to 2000 (0x7D0)

Response PDU

Function Code	1 byte	01(0x01)
Byte Count	1 byte	N*
Coil Status	n bytes	n = N or N+1

*N = Quantity of Outputs / 8, if the remainder is different than 0 then N = N+1

Here is an example of a request to read coils 20û38:

Request	hex	Response	hex
Function Code	01	Function Code	01
Starting Address Hi	00	Byte Count	03
Starting Address Lo	13	Coil Status 27-20	CD
No. of Coils Hi	00	Coil Status 35-28	6B
No. of Coils Lo	13	Coil Status 38-36	05

The status of outputs 27û20 is shown as the byte value CD hex, or binary 1100 1101. Output 27 is the MSB of this byte, and output 20 is the LSB.

By convention, bits within a byte are shown with the MSB to the left, and the LSB to the right. Thus the outputs in the first byte are æ27 through 20Æ, from left to right. The next byte has outputs æ35 through 28Æ, left to right. As the bits are transmitted serially, they flow from LSB to MSB: 20 . . . 27, 28 . . . 35, and so on.

In the last data byte, the status of outputs 38-36 is shown as the byte value 05 hex, or binary 0000 0101. Output 38 is in the sixth bit position from the left, and output 36 is the LSB of this byte. The five remaining high order bits are zero filled.

4.2 Read Discrete Inputs 02 (0x02)

This function code is used to read from 1 to 2000 contiguous status of discrete inputs in a remote device. The Request PDU specifies the starting address, ie the address of the first input specified, and the number of inputs. Inputs are addressed starting at zero. Therefore inputs 1-16 are addressed as 0-15.

The status of discrete inputs in the response message are packed as one input per bit of the data field. Status is indicated as 1= ON; 0= OFF. The LSB of the first data byte contains the input addressed in the query. The other inputs follow toward the high order end of this byte, and from low order to high order in subsequent bytes.

If the returned input quantity is not a multiple of eight, the remaining bits in the final data byte will be padded with zeros (toward the high order end of the byte). The Byte Count field specifies the quantity of complete bytes of data.

Request PDU

Function Code	1 byte	02 (0x02)
Starting Address	2 bytes	0x0000 to 0xFFFF
No. of Inputs	2 bytes	1 to 2000 (0x7D0)

Response PDU

Function Code	1 byte	01(0x01)
Byte Count	1 byte	N*
Input Status	n bytes	n = N or N+1

*N = Quantity of Outputs / 8, if the remainder is greater than of 0 then N = N+1

Here is an example of a request to read inputs 197 - 218:

Request	hex	Response	hex
Function Code	02	Function Code	02
Starting Address Hi	00	Byte Count	03
Starting Address Lo	C5	Input Status 204-197	AC
No. of Inputs Hi	00	Input Status 212-205	DB
No. of Inputs Lo	16	Input Status 218-213	35

The status of inputs 204û197 is shown as the byte value AC hex, or binary 1010 1100. Input 204 is the MSB of this byte, and input 197 is the LSB.

The status of inputs 218û213 is shown as the byte value 35 hex, or binary 0011 0101. Input 218 is in the third bit position from the left, and input 213 is the LSB. The two remaining high order bits are zeo filled.

4.3 Read Holding Registers 03 (0x03)

This function code is used to read the contents of a contiguous block of holding registers in a remote device. The Request PDU specifies the starting register address and the number of registers. Registers are addressed starting at zero. Therefore registers 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request PDU

Function Code	1 byte	03 (0x03)
Starting Address	2 bytes	0x0000 to 0xFFFF
No. Holdings Registers	2 bytes	1 to approx 125 (0x7D)

Response PDU

Function Code	1 byte	03(0x03)
Byte Count	1 byte	2 x N*
Register Values	n bytes	n = 2 x N

*N = Number of Registers

Here is an example of a request to read registers 108-110:

Request	hex	Response	hex

Function Code	03	Function Code	03
Starting Address Hi	00	Byte Count	06
Starting Address Lo	6B	Register Value Hi (108)	02
No. of Registers Hi	00	Register Value Lo (108)	2B
No. of Registers Lo	03	Register Value Hi (109)	00
		Register Value Lo (109)	
00		Register Value Hi (110)	00
		Register Value Lo (110)	64

The contents of register 108 are shown as the two byte values of 02 2B hex, or 555 decimal. The contents of registers 109û110 are 00 00 and 00 **64 hex, or 0 and 100 decimal, respectively.**

4.4 Read Input Registers 04 (0x04)

This function code is used to read from 1 to approx. 125 contiguous input registers in a remote device. The Request PDU specifies the starting register address and the number of registers. Registers are addressed starting at zero. Therefore input registers 1-16 are addressed as 0-15.

The register data in the response message are packed as two bytes per register, with the binary contents right justified within each byte. For each register, the first byte contains the high order bits and the second contains the low order bits.

Request PDU

Function Code	1 byte	04 (0x04)
Starting Address	2 bytes	0x0000 to 0xFFFF
No. Input Registers	2 bytes	1 to approx 125 (0x7D)

Response PDU

Function Code	1 byte	04(0x04)
Byte Count	1 byte	2 x N*
Input Register Values	n bytes	n = 2 x N

*N = Number of Registers

Here is an example of a request to read input register 9:

Request	hex	Response	hex
Function Code	04	Function Code	04
Starting Address Hi	00	Byte Count	02
Starting Address Lo	08	Value Register 9 Hi	00

No. Input Registers Hi	00	Value Register 9 Lo	0A
No. Input Registers Lo	01		

The contents of input register 9 are shown as the two byte values of 00 0A h ex, or 10 decimal.

4.5 Force Single Coil 05 (0x05)

This function code is used to force a single coil to either ON or OFF in a remote device.

The requested ON/OFF state is specified by a constant in the request data field. A value of FF 00 hex requests the output to be ON. A value of 00 00 requests it to be OFF. All other values are illegal and will not affect the output.

The Request PDU specifies the address of the coil to be forced. Coils are addressed starting at zero. Therefore coil 1 is addressed as 0. The requested ON/OFF state is specified by a constant in the Coil Value field. A value of 0xFF00 requests the coil to be ON. A value of 0x0000 requests the coil to be off. All other values are illegal and will not affect the coil.

The normal response is an echo of the request that is returned after the coil state has been set.

Request PDU

Function Code	1 byte	05 (0x05)
Coil Address	2 bytes	0x0000 to 0xFFFF
Coil Value	2 bytes	0x0000 or 0xFF00

Response PDU

Function Code	1 byte	05 (0x05)
Coil Address	2 bytes	0x0000 to 0xFFFF
Coil Value	2 bytes	0x0000 or 0xFF00

Here is an example of a request to force coil 173 ON:

Request	hex	Response	hex
Function Code	05	Function Code	05
Coil Address Hi	00	Coil Address Hi	00
Coil Address Lo	AC	Coil Address Lo	AC
Coil Value Hi	FF	Coil Value Hi	FF
Coil Value Lo	00	Coil Value Lo	00

4.6 Write Single Register 06 (0x06)

This function code is used to write a single holding register in a remote device.

The Request PDU specifies the address of the register to be written. Registers are addressed starting at zero. Therefore register 1 is addressed as 0.

The normal response is an echo of the request, returned after the register contents have been written.

Request PDU

Function Code	1 byte	06 (0x06)
Register Address	2 bytes	0x0000 to 0xFFFF
Register Value	2 bytes	0x0000 to 0xFFFF

Response PDU

Function Code	1 byte	06 (0x06)
Register Address	2 bytes	0x0000 to 0xFFFF
Register Value	2 bytes	0x0000 to 0xFFFF

Here is an example of a request to write register 2 to 0x0003.

Request	hex	Response	hex
Function Code	06	Function Code	06
Register Address Hi	00	Register Address Hi	00
Register Address Lo	01	Register Address Lo	01
Register Value Hi	00	Register Value Hi	00
Register Value Lo	03	Register Value Lo	03

4.7 Force Multiple Coils 15 (0x0F)

This function code is used to force each coil in a sequence of coils to either ON or OFF in a remote device. The Request PDU specifies the coil references to be forced. Coils are addressed starting at zero. Therefore coil 1 is addressed as 0.

The requested ON/OFF states are specified by contents of the request data field. A logical '1' in a bit position of the field requests the corresponding output to be ON. A logical '0' requests it to be OFF.

The normal response returns the function code, starting address, and

quantity of coils forced.

Request PDU

Function Code	1 byte	15 (0x0F)
Starting Address	2 bytes	0x0000 to 0xFFFF
Quantity of coils	2 bytes	0x0001 to 0x07B0
Byte Count	1 byte	n
Output value	n x 1 byte	value

Response PDU

Function Code	1 byte	15 (0x0F)
Starting Address	2 bytes	0x0000 to 0xFFFF
Quantity of coilsq	2 bytes	0x0001 to 0x07B0

Here is an example of a request to force a series of 10 coils starting at coil 20:

The request data contents are two bytes: CD 01 hex (1100 1101 0000 0001 binary). The binary bits correspond to the outputs are:

Bit :	1	1	0	0	1	1	0	1	0	0	0	0	0	0	0	1
Coil:	27	26	25	24	23	22	21	20	û	û	û	û	û	û	29	28

The first byte transmitted (CD hex) addresses outputs 27-20, with the least significant bit addressing the lowest output (20) in this set. The next byte transmitted (01 hex) addresses outputs 29-28, with the least significant bit addressing the lowest output (28) in this set. Unused bits in the last data byte should be zeroûfilled.

Request	hex	Response	hex
Function Code	0F	Function Code	0F
Coil Address Hi	00	Coil Address Hi	00
Coil Address Lo	13	Coil Address Lo	13
Quantity of Coils Hi	00	Quantity of Coil Hi	00
Quantity of Coils Lo	0A	Quantity of Coil Lo	0A
Byte Count	02		
Outputs Value Hi	CD		
Outputs Value Lo	01		

4.8 Write Multiple Registers 16 (0x10)

This function code is used to write a block of contiguous registers (1 to approx. 120 registers) in a remote device.

The requested written values are specified in the request data field. Data is packed as two bytes per register.

The normal response returns the function code, starting address, and quantity of registers written.

Request PDU

Function Code	1 byte	16 (0x10)
Starting Address	2 bytes	0x0000 to 0xFFFF
No. of registers	2 bytes	0x0001 to approx 0x0078
Byte Count	1 byte	2 x n
Registers value	n x 2 bytes	value

Response PDU

Function Code	1 byte	15 (0x0F)
Starting Address	2 bytes	0x0000 to 0xFFFF
No. of registers	2 bytes	0x0001 to approx 0x0078

Here is an example of a request to write two registers starting at 2 to 00 0A and 01 02 hex:

Request	hex	Response	hex
Function Code	10	Function Code	10
Starting Address Hi	00	Starting Address Hi	00
Starting Address Lo	01	Starting Address Lo	01
No. of Registers Hi	00	No. of Registers Hi	00
No. of Registers Lo	02	No. of Registers Lo	02
Byte Count	04		
Register Value Hi	00		
Register Value Lo	0A		
Register Value Hi	01		
Register Value Lo	02		

4.9 Read File Record 20 / 6 (0x14 / 0x06)

This function code is used to perform a file record read. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of the number of 16-bit words.

A file is an organization of records. Each file contains 10000 records, addressed 0000 to 9999 decimal or 0x0000 to 0x270F. For example, record 12 is addressed as 12.

The function can read multiple groups of references. The groups can be separated, ie non-contiguous, but the references within each group must be sequential.

Each group is defined in a separate æsub-requestÆ field that contains 7 bytes:

The Reference Type: 1 byte (must be specified as 6)
The File Number: 2 bytes (1 to 10, 0x0001 to 0x000A)
The Starting Record Number within the file: 2 bytes
The Length of the Record to be read: 2 bytes.

The quantity of words to be read, combined with all other fields in the expected response, must not exceed the allowable length of MODBUS messages: 256 bytes.

The normal response is a series of æsub-responsesÆ, one for each æsub-requestÆ. The byte count field is the total combined count of bytes in all æsub-responsesÆ. In addition, each æsub-responseÆ contains a field that shows its own byte count.

Request PDU

Function Code	1 byte	20 (0x14)
Request Data Length	1 byte	0x07 to 0xF5 bytes
Reference Type, SubReq x	1 byte	6
File Number, SubReq x	2 bytes	0x0000 to 0xFFFF
Record Number, SubReq x	2 bytes	0x0000 to 0xFFFF
Record Length, SubReq x	2 bytes	0x0000 to 0xFFFF words
Reference Type, SubReq x+1	1 byte	6
.....		

Response PDU

Function Code	1 byte	20(0x14)
Response Data Length	1 byte	0x07 to 0xF5 bytes
File Response Length, SubReq x	1 byte	0x07 to 0xF5 bytes
Reference Type, SubReq x	1 byte	6
Record Data, SubReq x	Nx2 bytes	
File Response Length,SubReqx+1	1 byte	
Reference Type, SubReq x+1	1 byte	6
.....		

Here is an example of a request to read one file record from a remote device:

it consists of two words from file 4, starting at record 1 (address 0001).

Request	hex	Response	hex

Function Code	14	Function Code	10
Request Data Length	07	Response Data Length	06
Reference Type	06	File Response Length	05
File Number Hi	00	Reference Type	06
File Number Lo	04	Record Data Hi	0D
Record Number Hi	00	Record Data Lo	FE
Record Number Lo	01	Record Data Hi	00
Record Length Hi	00	Record Data Lo	20
Record Length Lo	02		

4.10 Write File Record 21 / 6 (0x15 / 0x06)

This function code is used to perform a file record write. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of the number of 16-bit words.

A file is an organization of records. Each file contains 10000 records, addressed 0000 to 9999 decimal or 0x0000 to 0x270F. For example, record 12 is addressed as 12.

The function can write multiple groups of references. The groups can be separate, ie nonûcontiguous, but the references within each group must be sequential.

Each group is defined in a separate æsub-requestÆ field that contains 7 bytes plus the data:

- The Reference Type: 1 byte (must be specified as 6)
- The File Number: 2 bytes (1 to 10, hex 0001 to 000A)
- The Starting Record Number within the file: 2 bytes
- The Length of the Record to be written: 2 bytes
- The Data to be written: 2 bytes per register.

The quantity of words to be written, combined with all other fields in the request, must not exceed the allowable length of MODBUS messages: 256 bytes.

The normal response is an echo of the request.

Request PDU

Function Code	1 byte	21 (0x15)
Request Data Length	1 byte	0x07 to 0xF5 bytes
Reference Type, SubReq x	1 byte	6
File Number, SubReq x	2 bytes	0x0000 to 0xFFFF
Record Number, SubReq x	2 bytes	0x0000 to 0xFFFF
Record Length, SubReq x	2 bytes	0x0000 to 0xFFFF words

Record Data, SubReq x	n x 2 bytes	
Reference Type, SubReq x+1	1 byte	6
.....		

Response PDU

Function Code	1 byte	21 (0x15)
Response Data Length	1 byte	0x07 to 0xF5 bytes
Reference Type, SubReq x	1 byte	6
File Number, SubReq x	2 bytes	0x0000 to 0xFFFF
Record Number, SubReq x	2 bytes	0x0000 to 0xFFFF
Record Length, SubReq x	2 bytes	0x0000 to 0xFFFF words
Record Data, SubReq x	n x 2 bytes	
Reference Type, SubReq x+1	1 byte	6
.....		

Here is an example of a request to write one file record to a remote device:

The group consists of three words in file 4, starting at record 7 (address 0007).

Request	hex	Response	hex
Function Code	15	Function Code	15
Request Data Length	0D	Response Data Length	0D
Reference Type	06	Reference Type	06
File Number Hi	00	File Number Hi	00
File Number Lo	04	File Number Lo	04
Record Number Hi	00	Record Number Hi	00
Record Number Lo	07	Record Number Lo	07
Record Length Hi	00	Record Length Hi	00
Record Length Lo	03	Record Length Lo	03
Record Data Hi	06	Record Data Hi	06
Record Data Lo	01	Record Data Lo	01
Record Data Hi	03	Record Data Hi	03
Record Data Lo	02	Record Data Lo	02
Record Data Hi	04	Record Data Hi	04
Record Data Lo	01	Record Data Lo	01

4.11 Mask Write Register 22 (0x16)

This function code is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the holding register's current contents. The function can be used to set or clear individual bits in the register.

The request specifies the holding register to be written, the data to

be used as the AND mask, and the data to be used as the OR mask. Registers are addressed starting at zero. Therefore registers 1-16 are addressed as 0-15.

The functionÆs algorithm is:
Result = (Current Contents AND And_Mask) OR (Or_Mask AND And_Mask)

For example:

	Hex	Binary
Current Contents =	12	0001 0010
And_Mask =	F2	1111 0010
Or_Mask =	25	0010 0101
And_Mask =	0D	0000 1101
Result =	17	0001 0111

Note that if the Or_Mask value is zero, the result is simply the logical ANDing of the current contents and And_Mask. If the And_Mask value is zero, the result is equal to the Or_Mask value. The contents of the register can be read with the Read Holding Register function (function code 03).

The normal response is an echo of the request. The response is returned after the register has been written.

Request PDU

Function Code	1 byte	22 (0x16)
Reference Address	2 bytes	0x0000 to 0xFFFF
And_Mask	2 bytes	0x0000 to 0xFFFF
Or_Mask	2 bytes	0x0000 to 0xFFFF

Response PDU

Function Code	1 byte	22 (0x16)
Reference Address	2 bytes	0x0000 to 0xFFFF
And_Mask	2 bytes	0x0000 to 0xFFFF
Or_Mask	2 bytes	0x0000 to 0xFFFF

Here is an example of a Mask Write to register 5 in a remote device, using the above mask values.

Request	hex	Response	hex
Function Code	16	Function Code	16
Ref. Address Hi	00	Ref. Address Hi	00
Ref. Address Lo	04	Ref. Address Lo	04
And_Mask Hi	00	And_Mask Hi	00

And_Mask Lo	F2	And_Mask Lo	F2
Or_Mask Hi	00	Or_Mask Hi	00
Or_Mask Lo	25	Or_Mask Lo	25

4.12 Read/Write Holding Registers 23 (0x17)

This function code performs a combination of one read operation and one write operation in a single MODBUS transaction.

Holding registers are addressed starting at zero. Therefore holding registers 1-16 are addressed as 0-15.

The request specifies the starting address and number of holding registers to be read as well as the starting address, number of holding registers, and the data to be written. The byte count specifies the number of bytes to follow in the write data field.

The normal response contains the data from the group of registers that were read. The byte count field specifies the quantity of bytes to follow in the read data field.

Request PDU

Function Code	1 byte	23 (0x17)
Read Starting Address	2 bytes	0x0000 to 0xFFFF
Quantity to read	2 bytes	0x0001 to approx 0x76
Write Starting Address	2 bytes	0x0000 to 0xFFFF
Quantity to write	2 bytes	0x0001 to approx 0x76
Write byte count	1 byte	2 x N*
Write registers value	N* x 2 bytes	

Response PDU

Function Code	1 byte	03(0x03)
Byte Count	1 byte	2 x N*
Read Register Values	N* x 2 bytes	

*N = Number of Registers

Here is an example of a request to read six registers starting at register 4, and to write three registers starting at register 15:

Request	hex	Response	hex
Function Code	17	Function Code	17

Read Starting Addr Hi	00	Byte Count	0C
Read Starting Addr Lo	04	Register 4 Data Hi	02
Quantity to Read Hi	00	Register 4 Data Lo	2B
Quantity to Read Lo	06	Register 5 Data Hi	00
Write Starting Addr Hi	00	Register 5 Data Lo	00
Write Starting Addr Lo	0F	Register 6 Data Hi	00
Quantity to Write Hi	00	Register 6 Data Lo	64
Quantity to Write Lo	03	Register 7 Data Hi	00
Write Byte count	06	Register 7 Data Lo	54
Write Reg15 Data Hi	00	Register 8 Data Hi	01
Write Reg 15 Data Lo	FF	Register 8 Data Lo	02
Write Reg 16 Data Hi	00	Register 9 Data Hi	01
Write Reg16 Data Lo	FF	Register 9 Data Lo	03
Write Reg 17 Data Hi	00		
Write Reg 17 Data Lo	FF		

4.13 Read Device Identification 43/14 (0x2B / 0E)

This function code allows reading the identification and additional information relative to the physical and functional description of a remote device.

The Read Device Identification interface is modeled as an address space composed of a set of addressable data elements. The data elements are called objects and an object Id identifies them.

The interface consists of 3 categories of objects :

Basic Device Identification.
All objects of this category are mandatory, ie
VendorName, Product Code, and Revision Number.

Regular Device Identification.
In addition to Basic Device Identification, the device provides additional and optional identification and description data objects. All of the objects of this category are defined in the standard but their implementation is optional.

Extended Device Identification.
In addition to Regular Device Identification, the device provides additional and optional identification and description of private data. All of these data are device dependent.

Object Id	Object Name	Type	M/O	Category
0x00	VendorName	ASCIIString	Mandatory	Basic
0x01	ProductCode	ASCIIString	Mandatory	Basic

0x02	MajorMinorRevision	ASCIIString	Mandatory	Basic
0x03	VendorUrl	ASCIIString	Optional	Regular
0x03	ProductName	ASCIIString	Optional	Regular
0x04	VendorUrl	ASCIIString	Optional	Regular
0x05	UserApplicationName	ASCIIString	Optional	Regular
0x07 .. 0x7F	Reserved			
0x80 .. 0xFF	Product dependent		Optional	Extended

A Modbus Encapsulated Interface assigned number 14 identifies the Read Identification request. Four access types are defined:

01 : request to get the basic device identification (stream access)
02 : request to get the regular device identification (stream access)
03 : request to get the extended device identification (stream access)
04 : request to get one specific identification object (individual access)

In the case where the identification data does not fit into a single response, several request/response transactions may be required. The Object Id byte gives the identification of the first object to obtain. For the first transaction, the client must set the Object Id to 0 to obtain the start of the device identification data. For the following transactions, the client must set the Object Id to the value returned by the server in its previous response.

If the Object Id does not match any known object, the server responds as if object 0 were pointed out (restart at the beginning).

In case of an individual access: ReadDevId code 04, the Object Id in the request gives the identification of the object to obtain.

If the Object Id does not match any known object, the server returns an exception response with exception code = 02 (Illegal data address).

The normal response contains the following information :

ReadDevId Code	Same as request ReadDevId code : 01, 02, 03 or 04
Conformity Level	Identification conformity level of the device and type of supported access 01 : basic identification (stream access only) 02 : regular identification (stream access only) 03 : extended identification (stream access only) 81 : basic identification (stream access and

	individual access) 82 : regular identification (stream access and individual access) 83 : extended identification (stream access and individual access)
More Follows	In case of ReadDevId codes 01, 02 or 03 (stream access), If the identification data doesn't fit into a single response, several request/response transactions may be required. 00 : no more Object are available FF : other identification Object are available and further Modbus transactions are required. In case of ReadDevId code 04 (individual access), this field must be set to 00.
Next Object Id	If "MoreFollows = FF", identification of the next Object to be asked for. if "MoreFollows = 00", must be set to 00 (useless)
Number Of Objects	Number of identification Object returned in the response (for an individual access, Number Of Objects = 1)
Object0.Id	Identification of the first Object returned in the PDU (stream access) or the requested Object (individual access)
Object0.Length	Length of the first Object in bytes
Object0.Value	Value of the first Object (Object0.Length bytes)
...	
ObjectN.Id	Identification of the last Object (within the response)
ObjectN.Length	Length of the last Object in bytes
ObjectN.Value	Value of the last Object (ObjectN.Length bytes)

Request PDU

Function Code	1 byte	43 (0x2B)
MEI Type	1 byte	0x0E
Read ID code	1 byte	01/02/03/04
Object ID	1 byte	0x00 to 0xFF

Response PDU

Function Code	1 byte	43 (0x2B)
MEI Type	1 byte	0x0E
Read ID code	1 byte	01/02/03/04
Conformity level	1 byte	
More follows	1 byte	00 / FF
Next Object Id	1 byte	

Number of objects	1 byte
Object ID x	1 byte
Object length x	1 byte
Object value x	1 byte
Object ID x+1	1 byte
Object length x+1	1 byte
Object value x+1	1 byte
.....	

Example of a Read Device Identification request for "Basic device identification". In this example all information are sent in one response PDU.

Request	hex	Response	hex
Function Code	2B	Function Code	2B
MEI Type	0E	MEI Type	0E
Read Id Code	01	Register Id code	01
Object Id	00	Conformity level	01
		More follows	00
		Next Object Id	00
		Number of object	03
		Object ID	00
		Object Length	16
		Object Value	"company identification"
		Object ID	01
		Object Length	0C
		Object Value	"product code"
		Object ID	02
		Object Length	07
		Object Value	"version"

5. Acknowledgments

This document is the result of a collaboration of members of the MODBUS Community located at www.modbus.org.

6. Security Considerations

This RFC does not discuss security issues and is not believed to raise any security issues not already endemic to MODBUS communications. Since MODBUS/TCP is based on TCP/IP, it is not inherently secure.

7. References

[RFC 791] Internet Protocol, Sep81 DARpa

8. Authors' Addresses

Dennis Dub
Schneider Automation Inc.
1 High St.
N. Andover, MA 01845-2699
USA

Phone : 978-975-2891
Email : dennis.dube@modicon.com

Jacques Camerini
Schneider Automation S.A.
245, route des Lucioles B.P
147 F-06903
Sophia-Antipolis, France

Phone : 33 4 92 38 2045
Email : jacques.camerini@modicon.com

APPENDIX A - Full Copyright Statement

This Internet-Draft will expire on November 10, 2002.