# Forlong MODBUS Protocol Specification

V1.1

2009 / 05

# MODBUS APPLICATION PROTOCOL SPECIFICATION

V1.1

CONTENTS

# Part I

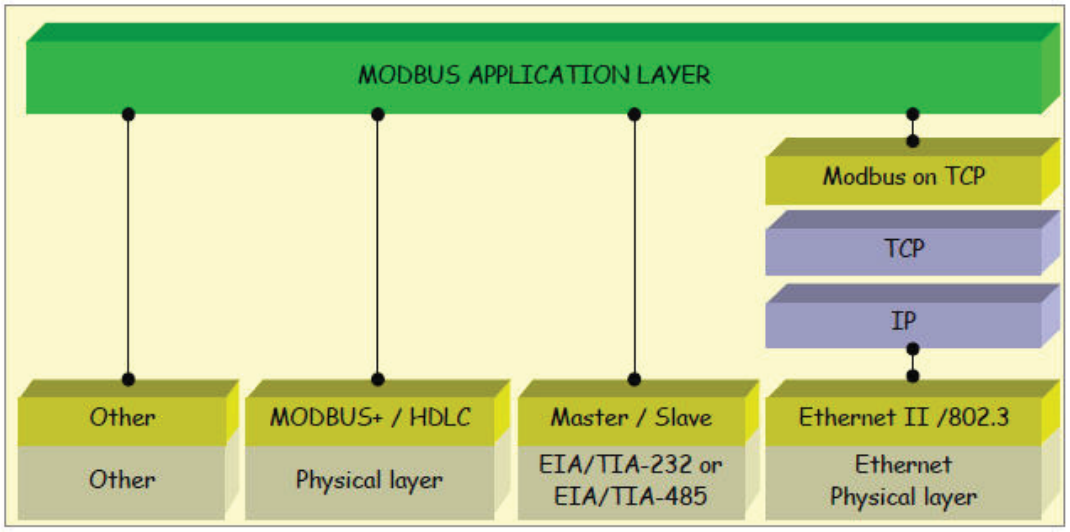
## Introduction

## 1 Scope of this document

This document provides information for forlong devices implementing the MODBUS RTU protocol.

MODBUS is an application layer messaging protocol, positioned at level 7 of the OSI model, that provides client/server communication between devices connected on different types of buses or networks. It is currently implemented using:

- TCP/IP over Ethernet. See MODBUS Messaging Implementation Guide V1.0a
- Asynchronous serial transmission over a variety of media (wire : EIA/TIA-232-E, EIA-422, EIA/TIA-485-A, fiber, radio, etc.)
- MODBUS PLUS, a high speed token passing network.




MODBUS Communication stack

The industry's serial de facto standard since 1979, MODBUS continue to enable millions of automation devices to communicate. Today support for the simple and elegant structure of MODBUS continues to grow. The Internet community can access MODBUS at a reserved system port 502 on the TCP/IP stack.

MODBUS is a request/reply protocol and offers services specified by function codes. MODBUS function codes are elements of MODBUS request/reply PDUs. The objectives of this document is to describe the function codes used within the framework of MODBUS transactions.

## 2 Protocol overview

For a detailed description of the MODBUS protocol please view the web site

www.modbus.org where the latest specs can be found.

# Part II

## 2 MODBUS transmission Modes

One serial transmission modes is defined: The RTU mode.

### 2.1 RTU Transmission mode

When devices communicate on a MODBUS serial line using the RTU (Remote Terminal Unit)mode, each 8-bit byte in a message contains two 4-bit hexadecimal characters. Each message must be transmitted in a continuous stream of characters.

The format (11bits) for each byte in RTU mode is:

Coding System: 8-bit binary

Bits per Byte: 1 start bit

8 data bits, least significant bit sent first

1 bit for parity completion

1 stop bit

Even parity is required.

## 2.2 Frame Checking Field:

Cyclical Redundancy Checking(CRC)

### 2.2.1 Frame description

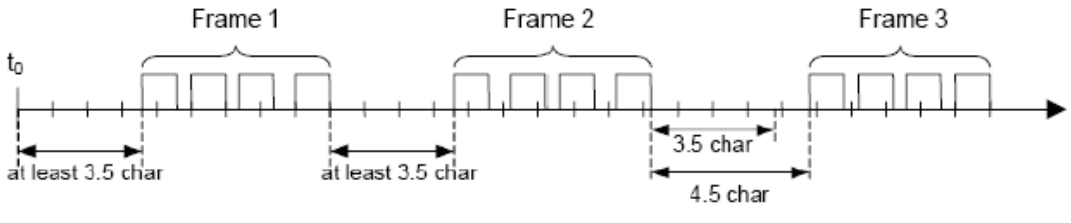
| Slave Address | Function Code | Data | CRC |
|---|---|---|---|
| 1byte | 1 byte | 0 up to 252 byte(s) | 2 bytes CRC Low CRCHi |

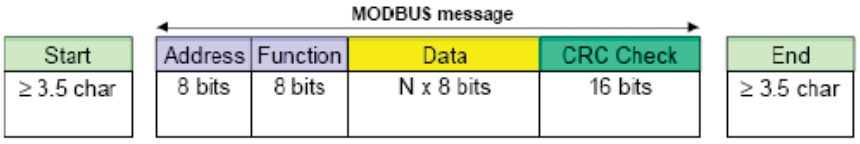
The maximum size of a MODBUS RTU frame is 256 bytes.

### 2.2.2 RTU Message Framing

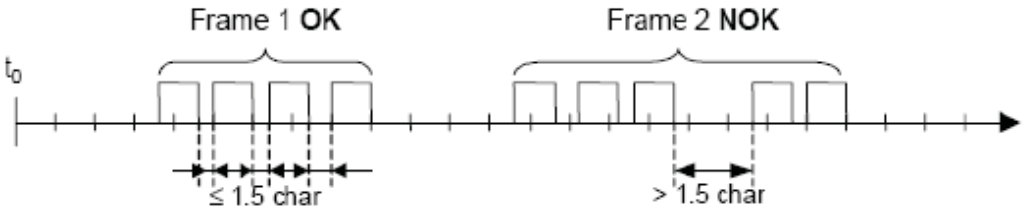
A MODBUS message is placed by the transmitting device into a frame that has a known beginning and ending point. This allows devices that receive a new frame to begin at the start of the message, and to know when the message is completed. Partial message must be detected and errors must be set as a result.

In RTU mode, message frames are separated by a silent interval of at least 3.5 character times. In the following sections, this time interval is called t3.5.



The entire message frame must be transmitted as a continuous stream of characters. If a silent interval of more than 1.5 character times occurs between two characters, the message frame is declared incomplete and should be discarded by the receiver.




**Note:**

The implementation of RTU reception driver may imply the message of a lot of interruptions due to the t1.5 and t3.5 times.

### 2.2.3 RTU CRC Checking

The RTU mode includes an error-checking field that is based on a Cyclical Redundancy Checking(CRC) method performed on the message contents.

### 2.2.4 Data signal Rate

forlong's slave device supports the following baud rates

| Baud Rate | Comments |
|---|---|
| 1200 | |
| 2400 | |
| 4800 | |
| 9600 | |

### 2.2.5 Data Formats

2.2.5.1 unsigned 16-bit integer word Format

The Modbus applications support 16 bit integer information for several of the function codes.

A read or write to a modbus register comprise a 2×8 bit byte.

2.2.5.2

Part 

## 3 MODBUS Function Codes

Forlong Modbus RTU uses a subset of the standard Modbus function codes to provide access to measurement and information registers. These standard function codes provides basic support for IEE32-bit floating point number, 16 bit integer .

| Function Code | Name | Usage |
|---|---|---|
| **0x04** | **Read Input Registers** | Used for reading floating point and 16 bit integer measurements |
| **Ox06** | **Write single Registers** | Used for writing floating point and 16 bit integer values to single registers |
| **0x10** | **Write multiple holding register** | Write multiple holding register |
| **0x11** | **Report Device ID** | Used for reading device information including device ID, description, software version etc |

### 3.1 04(0x04)Read Input Registers

This function codes is used to read 1 to 125 continue input registers in a remote device. The Request PDU specifies the starting register address and the number of register.

In the PDU Register are addressed starting at zero. Therefore input register numbered 1-16 are addressed as 0-15

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

**Request**

| | | |
|---|---|---|
| Function code | 1 Byte | 0X04 |
| Starting Address | 2 Byte | 0X0000 to 0XFFFF |
| Quantity of register | 2 Byte | 0x0001 to 0x007D |

**Response**

| | | |
|---|---|---|
| Function code | 1 Byte | 0X04 |
| Byte count | 1 Byte | 2 × N* |
| Register value | N* ×2 Bytes | |

N* = Quantity of registers

**Error**

| | | |
|---|---|---|
| Function code | 1 Byte | 0x84 |
| Exception code | 1 Byte | 0x01 or 0x02 or 0x03 or 0x04 |

An Example of a request to read input register 9 from slave address 2 using RTU format, where the register contains the 16 bit hex value 0x55AA

| Request | |
|---|---|
| Field Name | (Hex) |
| Slave Address | 02 |
| Function | 04 |
| Starting Address Hi | 00 |
| Starting Address Lo | 08 |
| No. of Register Hi | 00 |
| No. of Register Lo | 01 |
| Check Sum | CRC |
| Check Sum | CRC |

| Response | |
|---|---|
| Field Name | (Hex) |
| Slave Address | 02 |
| Function Code | 04 |
| Byte Count | 02 |
| Input register Hi | 55 |
| Input register Lo | AA |
| Check Sum | CRC |
| Check Sum | CRC |

### 3.2 06(0x06) Write Multiple register

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.

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

**Request**

| | | |
|---|---|---|
| Function code | 1 Byte | 0X06 |
| Register Address | 2 Byte | 0X0000 to 0XFFFF |
| Register Value | 2 Byte | 0x0000 to 0xFFFF |

**Response**

| | | |
|---|---|---|
| Function code | 1 Byte | 0X06 |
| Register Address | 2 Byte | 0x0000 to 0xFFFF |
| Register value | 2 Bytes | 0x0000 to 0xFFFF |

**Error**

| | | |
|---|---|---|
| Error code | 1 Byte | 0x86 |
| Exception code | 1 Byte | 0x01 or 0x02 or 0x03 or 0x04 |

**Example**

An Example of a writing to register 40001(Primary VT Ratio) the value 400,to slave address 5 in RTU mode

**Request**

| Request | |
|---|---|
| Field Name | (Hex) |
| Slave Address | 05 |
| Function | 06 |
| Register Address Hi | 00 |
| Register Address Lo | 00 |
| Register value Hi | 01 |
| Register value Lo | 90 |
| Check Sum | CRC |
| Check Sum | CRC |

| Response | |
|---|---|
| Field Name | (Hex) |
| Slave Address | 05 |
| Function Code | 06 |
| Register Address Hi | 00 |
| Register Address Lo | 00 |
| Register value Hi | 01 |
| Register value Lo | 90 |
| Check Sum | CRC |
| Check Sum | CRC |

### 3.3 16(0x10) Write Multiple register

This function code is used to write a block of contiguous 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**

| | | |
|---|---|---|
| Function code | 1 Byte | 0X10 |

| | | |
|---|---|---|
| Starting Address | 2 Byte | 0X0000 to 0Xffff |
| Quantity of register | 2 Byte | 0X0000 to 0XFFFF |
| Byte Count | 1 Byte | 2 × N* |
| Register value | N*×2 Byte | Value |

N* = Quantity of registers

**Response**

| | | |
|---|---|---|
| Function code | 1 Byte | 0X10 |
| Starting Address | 2 Byte | 0X0000 to 0Xffff |
| Quantity of register | 2 Bytes | 1 to 123 (0x7B) |

**Error**

| | | |
|---|---|---|
| Error code | 1 Byte | 0X90 |
| Exception Code | 1 Byte | 0x01 or 0x02 or 0x03 or 0x04 |

**Example**

An example of a writing to register 40915 (Pulse value for power) the value 1.0,to slave address 5 in RTU mode

| Request | |
|---|---|
| Field Name | (Hex) |
| Slave Address | 05 |
| Function code | 10 |
| Starting Address Hi | 03 |
| Starting Address Lo | 92 |
| No. of Register Hi | 00 |
| No. of Register Lo | 02 |
| Byte count | 04 |
| Register value Hi | 3F |
| Value | 80 |
| value | 00 |
| Register value Lo | 00 |
| Check Sum | 77 |
| Check Sum | 26 |

| Response | |
|---|---|
| Field Name | (Hex) |
| Slave Address | 05 |
| Function | 10 |

| | |
|---|---|
| Starting Address Hi | 03 |
| Starting Address Lo | 92 |
| No. of Register Hi | 00 |
| No. of Register Lo | 02 |
| Check Sum | E1 |
| Check Sum | E5 |

### 3.4 17(0x11) Report Device ID

This function code is used to read the description of the type. the current status, and other information .

The format of a normal response is shown in the following example. The data contents are specific to each type of device.

| | | |
|---|---|---|
| Function code | 1 Byte | 0X11 |

**Response**

| | | |
|---|---|---|
| Function Code | 1 byte | 11 |
| Byte count | 1 byte | 1A |
| Device ID | 1 byte | 0D |
| Run Indicator | 1 byte | FF<br>00=OFF<br>FF=ON |
| Description | 16 bytes | "D225 xxx.yy" |
| Serial number | 4 bytes | 0 to 4294967295 |
| Hardware Version Engine | 2 bytes | |
| Hardware Version Coms | 2 bytes | |
| Hardware Version Display | 2 bytes | |

**Error**

| | | |
|---|---|---|
| Error Code | 1 byte | 91 |
| Exception Code | 1 byte | 0x01 or 0x04 |

Example:

| | | |
|---|---|---|
| Slave Address | 1 Byte | 0X03 |
| Function | 1 Byte | 0X11 |
| Check Sum | 1 Byte | CRC |
| Check Sum | 1 Byte | CRC |

**Response**

| | | |
|---|---|---|
| Slave Address | 1 byte | 03 |
| Function Code | 1 byte | 11 |
| Byte count | 1 byte | 1A |
| Device ID | 1 byte | 0D |
| Run Indicator | 1 byte | FF |
| Description | 16 bytes | 44(“D”)D225 xxx.yy” |
| | | 32(“2”) |
| | | 32(“2”) |
| | | 35(“5”) |
| | | 20(“space “) |
| | | 30(“0”) |
| | | 30(“0) |
| | | 31(“1”) |
| | | 2E(“.”) |
| | | 30(“0) |
| | | 32(“2”) |
| | | 00 |
| | | 00 |
| | | 00 |
| | | 00 |
| | | 00 |
| Serial number Hi | 1 byte | 00 |
| Serial number Hi | 1 byte | 01 |
| Serial number Lo | 1 byte | E2 |
| Serial number Lo | 1 byte | 40 |
| software Version Engine | 1 bytes | 01 |
| Software Version Engine | 1 bytes | 02 |
| Software Version Coms | 1 bytes | 00 |
| Software Version Coms | 1 bytes | 00 |
| Software Version Display | 1 byte | 00 |
| Software Version Display | 1 byte | 00 |
| Check sum | 1 byte | CRC |
| Check sum | 1 byte | CRC |

{44 32 32 35 20 30 30 31 2E 30 32 00 00 00 00 00} - changeable if meter hardware is altered updated - see below

{00 01 E2 40} - Changeable per meter - Serial Number - see below

{01 02} - Changeable if meter software is changed - Software version number - see below

{00 00} - FIXED - Software Coms Version, Version of the Modbus protocol used - see below

{00 00} - FIXED at 0.0, Not used

{0D 62}

Description, i.e. "D225 001.01" which is the device name D225, and product revision state, i.e. 001.01 as an example.

If you made a minor change to the hardware, like component value change then this would be updated to 001.02, if you

change the electronics inside the meter, then this would be a major upgrade, so you would go from 001.02 to 2.00, then

any minor changes would be 002.01,002.02 etc

Software Engine version, this would be fixed in your software, but if you changed the software, then this value would be

updated, it is split in to major and minor updates, so first release would be 1.0 ( in two bytes, so 0x01 0x00) and

bug fix would be 1.1 etc (as two byes, 0x01 0x01, a major update to the code would cause the version number to from

1.1 to 2.0 (as two bytes 0x02 0x00) etc

Software Coms version, this is the same as the Software Engine version, but reflects what version of the modbus protocol it supports. set this to 6.2 ( 6 in one byte and 2 in the next byte, i.e. 0x06 0x02) to reflect you are using Autometers Modbus Protocol Specification V6.2.{xx}

Software Display Version. Set to 0.0 , not used on the D225

Changeable on each meter.

Serial Number. The serial number needs to be different on every meter. This would be set at the time of manufacture, and would not be changed by the user.

# Part IV

## 4 MODBUS Register map

This appendix describes all parameters accessible by Function Codes 0x04, 0x06,0x10,0x11.Parameters are grouped together according to the measurement been made, to simplify and speed up the reading of the data.

The availability of parameters and functions is depended on the device been accessed.

4.1 DRT-301C-Ⅱ register Map Overview

The following table describes the global register map for the function Codes 0X04(register read) and 0x10(register write) for DRT-301 C-Ⅱ

| Address (hex) | Length (bytes) | Parameter Name | Access (R/W) | Data Format | Units |
|---|---|---|---|---|---|
| 0x0010 | 4 | Voltage L1 | R | Float | V |
| 0x0012 | 4 | Voltage L2 | R | Float | V |
| 0x0014 | 4 | Voltage L3 | R | Float | V |
| | | | | | |
| 0x0030 | 4 | Voltage L1-L3 | R | Float | V |
| 0x0032 | 4 | Voltage L3-L2 | R | Float | V |
| 0x0034 | 4 | Voltage L2-L1 | R | Float | V |
| | | | | | |
| 0X004E | 4 | Frequency | R | Float | Hz |
| | | | | | |
| 0X0050 | 4 | Current L1 | R | Float | A |
| 0X0052 | 4 | Current L2 | R | Float | A |
| 0X0054 | 4 | Current L3 | R | Float | A |
| 0X0056 | 4 | Current Neutral | R | Float | A |
| 0X0058 | 4 | Current total | R | Float | A |
| | | | | | |
| 0x0090 | 4 | Power L1 | R | Float | kW |
| 0x0092 | 4 | Power L2 | R | Float | kW |
| 0x0094 | 4 | Power L3 | R | Float | kW |
| 0x0096 | 4 | Power Total | R | Float | kW |
| | | | | | |
| 0x00D0 | 4 | Apparent | R | Float | kVA |

| | | | | | |
|---|---|---|---|---|---|
| | | Power L1 | | | |
| 0x00D2 | 4 | Apparent Power L2 | R | Float | kVA |
| 0x00D4 | 4 | Apparent Power L3 | R | Float | kVA |
| 0x00D6 | 4 | Apparent Power Total | R | Float | kVA |
| | | | | | |
| 0x0110 | 4 | Reactive Power L1 | R | Float | kvar |
| 0x0112 | 4 | Reactive Power L2 | R | Float | kvar |
| 0x0114 | 4 | Reactive Power L3 | R | Float | kvar |
| 0x0116 | 4 | Reactive Power Total | R | Float | kvar |
| | | | | | |
| 0x0150 | 4 | Power Factor L1 | R | Float | |
| 0x0152 | 4 | Power Factor L2 | R | Float | |
| 0x0154 | 4 | Power Factor L3 | R | Float | |
| 0x0156 | 4 | Power Factor Total | R | Float | |
| | | | | | |
| 0x0160 | 4 | Import Energy | R | Float | KWh |
| 0x0166 | 4 | Export Energy | R | Float | KWh |
| 0X0618 | 4 | Total Energy | R | Float | kWh |
| | | | | | |
| 0x0164 | 4 | Import Reactive Energy | R | Float | Kvarh |
| 0x0168 | 4 | Export | R | Float | kvarh |

| | | | | | |
|---|---|---|---|---|---|
| | | Reactive Energy | | | |
| | | | | | |
| 0x0524 | 2 | Modbus slave address number | R/W | 16 bit | address |
| 0x0525 | 2 | Modbus slave Baud rate | R/W | 16 bit | 1200bps 0x04B0<br>2400bps 0x0960<br>4800bps 0x12C0<br>9600bps 0x2580 |
| 0X0526 | 4 | Serial number | R/W | Integer | (0001E240)h=(123456)D |
| | | | | | |
| 0x0550 | 2 | Meter Mode | R/W | 16bit | 0= 3 Phase 4 wire<br>1 = 3 Phase 3 wire<br>2 = 3 Phase 3 wire VT |
| | | | | | |

命令码解析:

正向有功电能: 0x01,0x04,0x01,0x60,0x00,0x02,0x70,0x29
反向有功电能: 0x01,0x04,0x01,0x66,0x00,0x02,0x90,0x28
总电能: 0x01,0x04,0x06,0x18,0x00,0x02,0xF1,0x44

Import Reactive Energy: 0x01,0x04,0x01,0x64,0x00,0x02,0x31,0xE8
Export Reactive Energy: 0x01,0x04,0x01,0x68,0x00,0x02,0xF1,0Xeb

A 相有功功率: 0x01,0x04,0x00,0x90,0x00,0x02,0x71,0xE6
B 相有功功率: 0x01,0x04,0x00,0x92,0x00,0x02,0xD0,0x26
C 相有功功率: 0x01,0x04,0x00,0x94,0x00,0x02,0x30,0x27
有功总功率: 0x01,0x04,0x00,0x96,0x00,0x02,0x91,0xE7

A 相无功功率: 0x01,0x04,0x01,0x10,0x00,0x02,0x71,0xF2
B 相无功功率: 0x01,0x04,0x01,0x12,0x00,0x02,0xD0,0x32
C 相无功功率: 0x01,0x04,0x01,0x14,0x00,0x02,0x30,0x33
无功总功率: 0x01,0x04,0x01,0x16,0x00,0x02,0x91,0xF3

A 相功率因数: 0x01,0x04,0x01,0x50,0x00,0x02,0x70,0x26
B 相功率因数: 0x01,0x04,0x01,0x52,0x00,0x02,0xD1,0xE6

C 相功率因数: 0x01,0x04,0x01,0x54,0x00,0x02,0x31,0xE7

总功率因数: 0x01,0x04,0x01,0x56,0x00,0x02,0x90,0x27

A 相视在功率: 0x01,0x04,0x00,0xD0,0x00,0x02,0x70,0x32
B 相视在功率: 0x01,0x04,0x00,0xD2,0x00,0x02,0xD1,0xF2
C 相视在功率: 0x01,0x04,0x00,0xD4,0x00,0x02,0x31,0xF3
总视在功率: 0x01,0x04,0x00,0xD6,0x00,0x02,0x90,0x33

A 相电压: 0x01,0x04,0x00,0x10,0x00,0x02,0x70,0x0E
B 相电压: 0x01,0x04,0x00,0x12,0x00,0x02,0xD1,0xCE
C 相电压: 0x01,0x04,0x00,0x14,0x00,0x02,0x31,0Xcf

Voltage L1-L3 0x01,0x04,0x00,0x30,0x00,0x02,0x71,0xC4
Voltage L3-L2 0x01,0x04,0x00,0x32,0x00,0x02,0xD0,0x04
Voltage L2-L1 0x01,0x04,0x00,0x34,0x00,0x02,0x30,0x05

A 相电流: 0x01,0x04,0x00,0x50,0x00,0x02,0x71,0xDA
B 相电流: 0x01,0x04,0x00,0x52,0x00,0x02,0xD0,0x1A
C 相电流: 0x01,0x04,0x00,0x54,0x00,0x02,0x30,0x1B
N 相电流: 0x01,0x04,0x00,0x56,0x00,0x02,0x91,0xDB
总电流: 0x01,0x04,0x00,0x58,0x00,0x02,0xF0,0x18

频率: 0x01,0x04,0x00,0x4E,0x00,0x02,0x11,0xDC

读表号: 00 04 05 24 00 01 70 DC (00 广播地址)
to change address from modbos 01 to 02
01 06 05 24 00 02 48 CC
to change from from modbus 02 to 254
02 06 05 24 00 FE 48 BE
to change from modbus 02 to 128
02 06 05 24 00 80 C8 9E

SN (serial number )
Request:0x01,0x11,0xC0,0x2C
Reponses: 01 11 1A 0D FF 44 32 32 35 20 30 30 31 2E 30 32 00 00 00 00 00 00 01 E2 40 01 02 00 00 00 00 0D 62

{01}

{11}

{1A} - FIXED - bytecount

{0D} - FIXED - meter Type ID

{FF} - FIXED - Run indicator

{44 32 32 35 20 30 30 31 2E 30 32 00 00 00 00 00} - changeable if meter hardware is altered updated - see below

{00 01 E2 40} - Changeable per meter - Serial Number - see below

{01 02} - Changeable if meter software is changed - Software version number - see below

{00 00} - FIXED - Software Coms Version, Version of the Modbus protocol used - see below

{00 00} - FIXED at 0.0, Not used

{0D 62}

following are not changeable

byte count = 1A

Device ID = 0D (for this D225 meter)

Run Indicator = FF

Description, i.e. "D225 001.01" which is the device name D225, and product revision state, i.e. 001.01 as an example.

If you made a minor change to the hardware, like component value change then this would be updated to 001.02, if you

change the electronics inside the meter, then this would be a major upgrade, so you would go from 001.02 to 2.00, then

any minor changes would be 002.01,002.02 etc

Software Engine version, this would be fixed in your software, but if you changed the software, then this value would be

updated, it is split in to major and minor updates, so first release would be 1.0 ( in two bytes, so 0x01 0x00) and

bug fix would be 1.1 etc (as two byes, 0x01 0x01, a major update to the code would cause the version number to from

1.1 to 2.0 (as two bytes 0x02 0x00) etc

Software Coms version, this is the same as the Software Engine version, but reflects what version of the modbus protocol it supports. set this to 6.2 ( 6 in one byte and 2 in the next byte, i.e.

0x06 0x02) to reflect you are using Autometers Modbus Protocol Specification V6.2.{xx}

Software Display Version. Set to 0.0 , not used on the D225

Changeable on each meter.

Serial Number. The serial number needs to be different on every meter. This would be set at the time of manufacture, and would not be changed by the user.