ECE 444 / 544

SUPERVISORY CONTROL
AND CRITICAL
INFRASTRUCTURE SYSTEMS

SESSION no. 15
Numbering systems & Data Transfer

String of 0, 1 on the media

- copper wire

\[ \text{Binary: } 0, 1 \text{: 2 states} \]

one bit or binary digit: BIT

4 bits \( \rightarrow \) Nibble \[ 0110 \]
Eight Bits: Byte (Octet)

16 bits = 2 bytes (1 word)

32 bits = double word

Order of bits

LSB least significant bit
MSB most significant bit

Big Endian - sending most significant bit first
101, 1011
↑ ↑
*decimal*
13 11

CG or GC
/ 1
198 108

words

CG CG

198
50.688
Numbers

12 bit integers → 0 → 4095 unsigned
16 bit integer 0 - 65535 unsigned

Signed or unsigned

→ 1 bit is a sign bit to indicate positive or negative
MSB is sign bit
6.12 **16 (0x10) Write Multiple registers**

This function code is used to write a block of contiguous registers (1 to 123 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

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function code</td>
<td>1 Byte</td>
<td>0x10</td>
</tr>
<tr>
<td>Starting Address</td>
<td>2 Bytes</td>
<td>0x0000 to 0xFFFF</td>
</tr>
<tr>
<td>Quantity of Registers</td>
<td>2 Bytes</td>
<td>0x0001 to 0x007B</td>
</tr>
<tr>
<td>Byte Count</td>
<td>1 Byte</td>
<td>N x 2</td>
</tr>
<tr>
<td>Registers Value</td>
<td>N x 2 Bytes</td>
<td>N = Quantity of Registers</td>
</tr>
</tbody>
</table>

*Note: N = Quantity of Registers*

### Response

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function code</td>
<td>1 Byte</td>
<td>0x10</td>
</tr>
<tr>
<td>Starting Address</td>
<td>2 Bytes</td>
<td>0x0000 to 0xFFFF</td>
</tr>
<tr>
<td>Quantity of Registers</td>
<td>2 Bytes</td>
<td>1 to 123 (0x7B)</td>
</tr>
</tbody>
</table>

### Error

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error code</td>
<td>1 Byte</td>
<td>0x90</td>
</tr>
<tr>
<td>Exception code</td>
<td>1 Byte</td>
<td>01 or 02 or 03 or 04</td>
</tr>
</tbody>
</table>

Here is an example of a request to write two registers starting at 2 to 00 0A and 01 02 hex:
Once the request has been processed by a server, a MODBUS response using the adequate MODBUS server transaction is built. Depending on the result of the processing two types of response are built: A positive MODBUS response, and an exception code. The request function code is provided to indicate the reason of the error.
The MODBUS application data unit is built by the client that initiates a MODBUS transaction. The function indicates to the server what kind of action to perform. The MODBUS application protocol establishes the format of a request initiated by a client.

The function code field of a MODBUS data unit is coded in one byte. Valid codes are in the range of 1 ... 255 decimal (the range 128 – 255 is reserved and used for exception responses). When a message is sent from a Client to a Server device the function code field tells the server what kind of action to perform. Function code "0" is not valid.

Sub-function codes are added to some function codes to define multiple actions.

The data field of messages sent from a client to server devices contains additional information that the server uses to take the action defined by the function code. This can include items like discrete and register addresses, the quantity of items to be handled, and the count of actual data bytes in the field.

The data field may be nonexistent (of zero length) in certain kinds of requests, in this case the server does not require any additional information. The function code alone specifies the action.

If no error occurs related to the MODBUS function requested in a properly received MODBUS ADU the data field of a response from a server to a client contains the data requested. If an error related to the MODBUS function requested occurs, the field contains an exception code that the server application can use to determine the next action to be taken.

For example a client can read the ON / OFF states of a group of discrete outputs or inputs or it can read/write the data contents of a group of registers.

When the server responds to the client, it uses the function code field to indicate either a normal (error-free) response or that some kind of error occurred (called an exception response). For a normal response, the server simply echoes to the request the original function code.

---

**Figure 4:** MODBUS transaction (error free)

For an exception response, the server returns a code that is equivalent to the original function code from the request PDU with its most significant bit set to logic 1.

---

**Figure 5:** MODBUS transaction (exception response)
6.13  17 (0x11) Report Server ID (Serial Line only)

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

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

**Request**

<table>
<thead>
<tr>
<th>Function code</th>
<th>1 Byte</th>
<th>0x11</th>
</tr>
</thead>
</table>

**Response**

<table>
<thead>
<tr>
<th>Function code</th>
<th>1 Byte</th>
<th>0x11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte Count</td>
<td>1 Byte</td>
<td></td>
</tr>
<tr>
<td>Server ID</td>
<td>1 Byte</td>
<td>device specific</td>
</tr>
<tr>
<td>Run Indicator Status</td>
<td>1 Byte</td>
<td>0x00 = OFF, 0xFF = ON</td>
</tr>
<tr>
<td>Additional Data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Error**

<table>
<thead>
<tr>
<th>Error code</th>
<th>1 Byte</th>
<th>0x91</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception code</td>
<td>1 Byte</td>
<td>01 or 04</td>
</tr>
</tbody>
</table>

Here is an example of a request to report the ID and status:

April 26, 2012  http://www.modbus.org  31/50
Since MODBUS communication is sent over physical layer - we can also observe traffic.

- wireshark
- get tools to process/collect data
Advantages of MODBUS

- Easy to read
- Easy to troubleshoot
- Server/device addressing (slave)
- Data integrity checking built-in CRC
- Widely supported by vendors
Disadvantages of MODBUS

- Message starting point, stepping point etc are time based
- Lack of client device address
- Limitations in update time especially on serial channel

- Lack of security timestamps
- Lack of data quality logic
- No data without odd logic
- DNP overcomes some of the limitations of Modbus.

- Originally developed by Westronics, then Harris, now GE.

- IEC Working Group developed:
  - IEC 60870.
    - Wasn't completed yet.

  DNP was based on earlier that wasn't ratified in 1990.

1993 released to public domain.
- Supported by IEEE
  - works group as an IEEE standard
    - maintains & updates
      the standard
Advantages of DNP 3.

- Object oriented - gives more flexibility
  - supports time stamps
  - Data quality check built-in
  - Has connection/device monitoring
  - Data integrity check (CRC)
  - Has master & slave device addressing
  - Widely supported by vendors

- Message structure allows efficient communication & timing

Better utilization of limited bandwidth
Disadvantages

- buggier messages
- more free flowing = challenges for reading messages
- DNP has limited provisions for security (sterility to change)
- specific subsets
  - direct connected serial, RS485 etc
  - encryption
    encryption ≠ rather than TCP/IP
- not widely supported by vendors or adopted by users
- Capable to support 65520 addresses system (16 bit)

- Supports unsolicited responses

  - Good tool if used carefully

  - Especially if have slower comms

  - Become less important with higher speed comm