Altech Corp. MEXICO

UNIVERSAL MONITOR DO-1

FOR MODBUS

REGISTER 101

SIMPLE DEVICE MONITORING
FOR YOUR BUSINESS

V3/24

The current Modbus standards dictate the use of Client-Server device terminology in favor of Master-Slave; However, common practice is still to use Master-Slave terms when referring to connected Modbus devices. For ease of use / integration of the **DO-1** device we have kept the current terminology with future plans to update to the new standard as it becomes more common place.

For now: Client = Master , Server = Slave and vice versa.

The latest electronic version of this guide is available for download here:
https://do-1.altechmexico.com/en

Modbus register addresses, a case in itself...

Handling Modbus register addresses is often a tricky business

One of the biggest challenges when evaluating measured values is setting the Modbus registers for the respective measuring devices or sensors. Although there is a standard, unfortunately not every device manufacturer adheres to it or interprets it differently, which repeatedly leads to different and confusing documentation.

Here is some procedural help:

1. The basis for this is first of all the existence of a carefully established **RS-485** cable connection and the correct Modbus settings. It must be ensured that the measuring device or sensor is properly connected in the respective measuring circuit.

2. Familiarize yourself with the description of the Modbus registers and look for technical terms.
   Such as: Entity Number Register Address, Register Number, Object Type etc.

3. Disregard **gain** (multiplier) and **offset** (deviations) for the time being

4. *The following must then be observed for register numbers:*

   4.a) Do they have **5** or **6 digits**?
   Traditionally they have 5 digits, extended conventions have 6 digits.

   4.b) If the **last digit** of the first register number or address is a **zero**, this is a **zero-based sequence**.

   4.c) Is it a **register number** ('Entity Number') or a **register address** ('Entity Address')?
   Register numbers are a combination of register type and register address (e.g. 40001: the 4 stands for the register type 'Holding Register' and the 1 for the register address).

   4.d) Check the documentation for information on whether **binary or hexadecimal numbering** is specified. e.g. 4X0000 : (4) means holding register, (X) stands for hexadecimal and 0 in the last position for a zero-based register address sequence.

5. Select a register from which known results are to be expected, e.g: Voltage (110V or 220V), frequency (50Hz or 60Hz) or temperature (24 degrees) etc. This makes the register settings and subsequent evaluation much easier.

6. Note the terminology used to describe the register (object type), such as analog, digital, discrete, holding register, input register, coil, input, output, etc. This information refers to the register type (object type), e.g., coil, input, input register, or holding register.

   ***Add. note*:** Occasionally, the Modbus function codes are also used in the descriptions, e.g. codes 3 and 16 refer to a holding register.

7. The information on the **data type / data format** can be found in the register description or table.

8. Then set the sequence when reading the data ('big endian', unless otherwise specified). 80% of device manufacturers assume a reading sequence from left to right and from the place value for numbers where the largest value is on the left and the smallest value is on the right.

9. Save register details. If there are already several devices on the bus, it is **recommended** to set a **device and register delay** for the test phase. Approximately 100-300ms, this can of course be changed later.

10. Now connect the new measuring device or the new sensor or the new template to the corresponding Modbus and save them under 'Device settings'.

11. Test the newly entered register; Restart the process screen under the menu item System and process screen. Then select the new meter or sensor register under the corresponding Modbus.

12. If the measuring points appear regularly and logically regardless of the values of the Y-axis, it can be assumed that the register setting is correct. In this case, the **gain** (multiplier) must now be calculated and entered in the created register so that the Y-axis displays the expected value. **Offset** (deviation) in the register settings can then be used to make a more precise adjustment.

However, if **no sporadic or illogical measuring** points appear in the register evaluation, the cause is incorrect register settings. **Check and change the settings** again in accordance with points 4 to 9. Each adjustment can be tested again as described in points 11 and 12. Only after a successful register evaluation of a measuring device or sensor can further required register settings be made more quickly and easily, as the necessary settings are already available and can be applied.

Additional information on Modbus registers:

Entity number – Entity address

The term "entity number" could be understood as a unique identifier or number that identifies a specific entity or component in a system. This entity can be, for example, a device, a sensor, an actuator or any object that is present in a system.

The "entity address" refers to the specific address or location where the data or properties of that entity can be accessed or manipulated. In a system with many different entities, the addresses can be used to access specific information or functions of an entity.

It is important to note that the exact meaning of "entity number" and "entity address" may depend on the context and the specific application in which these terms are used. It is possible that these terms are used in a particular system, protocol or software framework to describe specific concepts or structures.

Functions in the Modbus communication protocol specification

0 - Coil: This refers to a Modbus function that is used to read and write discrete outputs. Coils are normally used for binary outputs that can take the value "0" or "1".

1 - Discrete Input: This function is used to query discrete inputs in a Modbus-enabled device. Discrete inputs are binary inputs that can assume the status "0" or "1", e.g. buttons or switches.

3 - Input Register: This function is used to retrieve the values of input registers in a Modbus device. Input registers usually contain analog or digital input values such as temperature, pressure or sensor values..

4 - Holding Register: The Holding Register function enables values to be read and written to so-called holding registers of a Modbus device. Holding registers are usually used to store data such as configuration settings, setpoints or other variables.

These functions offer various options for communicating with Modbus devices and exchanging information via different types of data.

Data formats

1. Float 32 Bit: A 32-bit floating point number that can represent both fractional and whole numbers with a limited value range and limited precision. It corresponds to the IEEE standard 754 for single-precision floating point numbers.

2. Float 64 Bit: A 64-bit floating-point number, also known as a double-precision floating-point number. It offers a larger value range and higher precision compared to float 32 bits. It also follows the IEEE standard 754 for double precision floating point numbers.

3. Integer 16 Bit: A 16-bit integer that can store whole numbers within a certain value range. It uses 16 bits (2 bytes) of memory and can represent values from -32,768 to 32,767.

4. Integer 32 Bit: A 32-bit integer that can store larger integers within a certain value range. It uses 32 bits (4 bytes) of memory and can represent values from -2,147,483,648 to 2,147,483,647.

5. Integer 64 Bit: A 64-bit integer that can store even larger integers within a certain value range. It uses 64 bits (8 bytes) of memory and can represent values from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807.

6. uInteger 16 Bit: An unsigned 16-bit integer that can only store positive integers within a certain value range. It uses 16 bits (2 bytes) of memory and can represent values from 0 to 65,535.

7. uInteger 32 Bit: An unsigned 32-bit integer that can store larger positive integers within a certain value range. It uses 32 bits (4 bytes) of memory and can represent values from 0 to 4,294,967,295.

8. uInteger 64 Bit: An unsigned 64-bit integer that can store even larger positive integers within a certain value range. It uses 64 bits (8 bytes) of memory and can represent values from 0 to 18,446,744,073,709,551,615.

***Note**: It is important to note that the specific value range and memory size of these data types can vary depending on the programming language or platform used.*

Endian formats

Different endian formats are important to ensure that data can be exchanged correctly between systems with different architectures.

1. Little Endian: With little endian, the least significant byte (smallest digit) of a multi-byte date is saved first. The successive bytes are stored in ascending order. This means that the byte with the lowest value is at the lowest memory address.

2. Big Endian: With Big Endian, the most significant byte (largest digit) of a multi-byte date is saved first. The successive bytes are stored in descending order. This means that the byte with the highest value is at the lowest memory address.

3. Little Endian Reversed: Little Endian Reversed is not a common notation. However, it could refer to a variant of little endian in which the order of the bytes is reversed. This means that the most significant byte is at the lowest memory address and the least significant byte is at the highest memory address.

4. Big Endian Reverse: Big Endian Reverse is also not a common notation. It could refer to a variant of big endian in which the order of the bytes is reversed. This means that the least significant byte is at the lowest memory address and the most significant byte is at the highest memory address.

Manufacturers are referring to various documentation variants, such as:

“**Register reading MSB to (→) LSB**” refers to Big Endian

“**Register reading LSB to (→) MSB**” refers to Little Endian

“0X, 1X, 3X, 4X” type is referring to register (object) type like coil, input, input register or holding register.

“4X0000” is referring to a holding register (4) , (X) for hexadecimal and to 0-(zero) based address register 0.

"Signed" or **“INT”** refers to a signed integer data type

"Unsigned" or **“USINT”** or **“UINT”** refers to a unsigned integer data type

"Unsigned short" or **“ UINT16”** refers 16-bit unsigned integer data type

“Short” refers to a 16-bit integer register type

“INT16” refers to a signed 16-bit integer data type

“1-Word” or **“2 Bytes”** refers to a 16 bit-register

“INT32” or **“DINT”** refers to a signed 32-bit integer data type

“UINT32” or **“UDINT”** refers to a unsigned 32-bit integer register type

“Long” refers to a 32-bit integer register type

"Unsigned long" refers to a 32-bit unsigned integer data type

“2-Word” or **“DWORD”** refers to 32-bits or two consecutive 16-bit registers

“Float” or **“Float32”** or **“FP32”** or **“REAL”** refers to 32-bits floating point register

“Double” or **“Float64”** or **“FP64”** or **“LREAL”** refers to 64-bits floating point register

“BOOL” refers to 1-bit register (0 or 1)

“HEX” refers to hexadecimal representation

“4-Word” or **“LWORD”** refers to 64-bits or four consecutive 16-bit registers