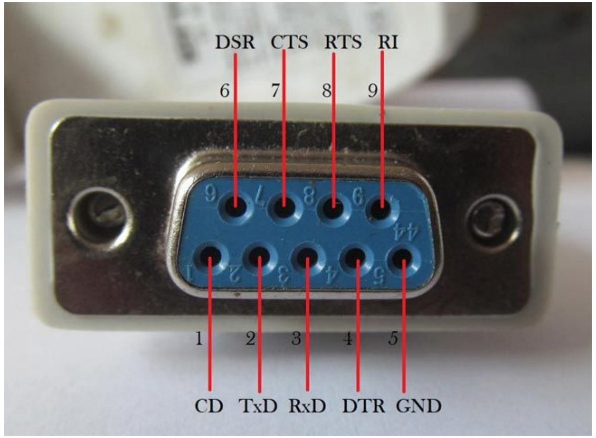
# SY486K MICS
# Lecture 7

## Modbus

March 2023

# Outline

- History
- Intro
- Object Types
- Protocols
- Message Format
  - Function Codes
  - Data Format
  - CRC
- Physical Medium
- Examples
- Modbus on AB micro820



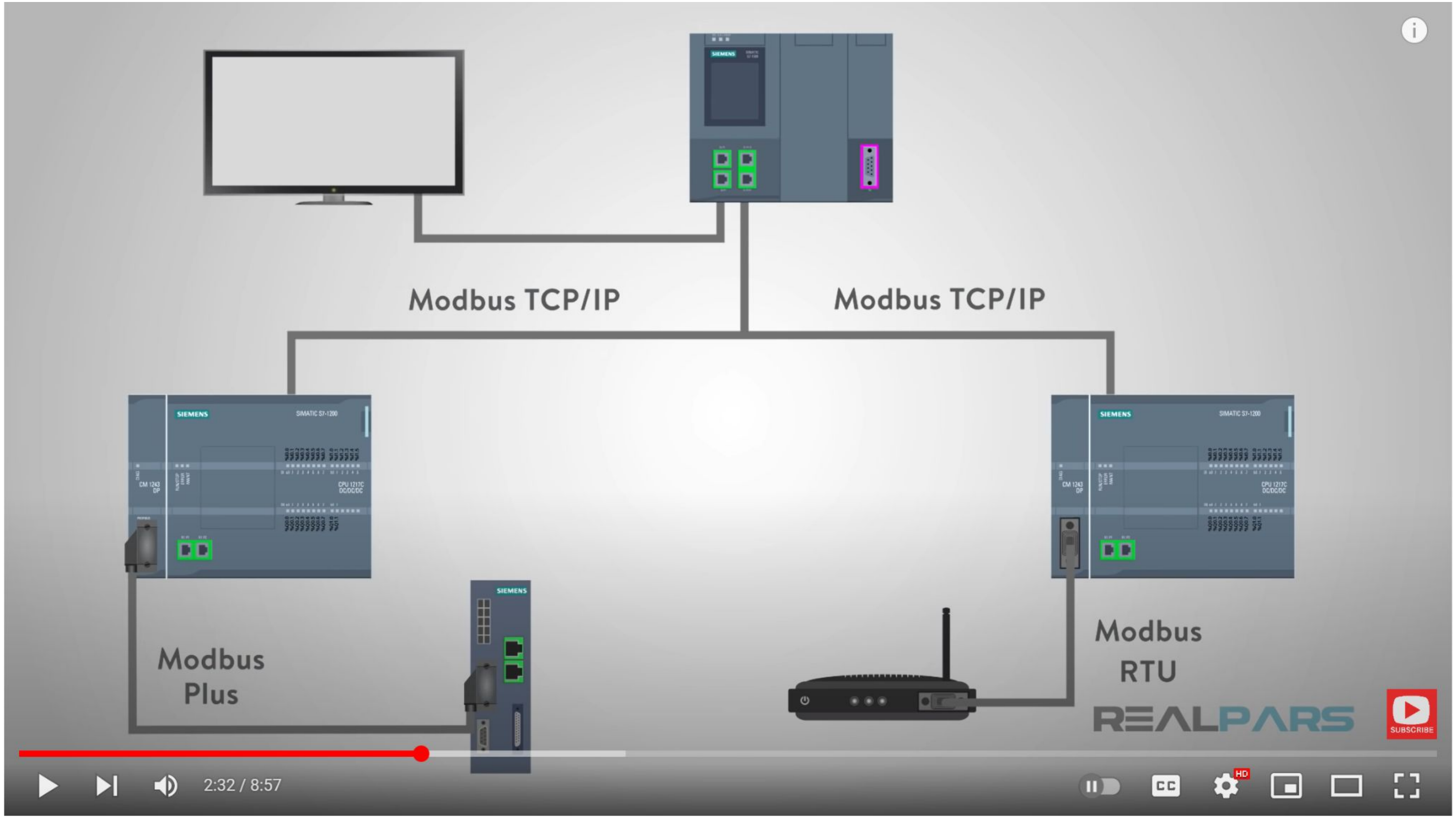

# Where did Modbus come from?

**Modbus** is a serial communication protocol developed by Modicon® in 1979 for use with its 084 programmable logic controllers (PLCs).

Modbus has become a de facto standard communication protocol and is now a commonly available means of connecting industrial electronic devices.

Modbus is popular in industrial environments because it is openly published and royalty-free. It was developed for industrial applications, is relatively easy to deploy and maintain compared to other standards, and places few restrictions on the format of the data to be transmitted.

https://www.youtube.com/watch?v=txi2p5_OjKU




What is Modbus and How does it Work?

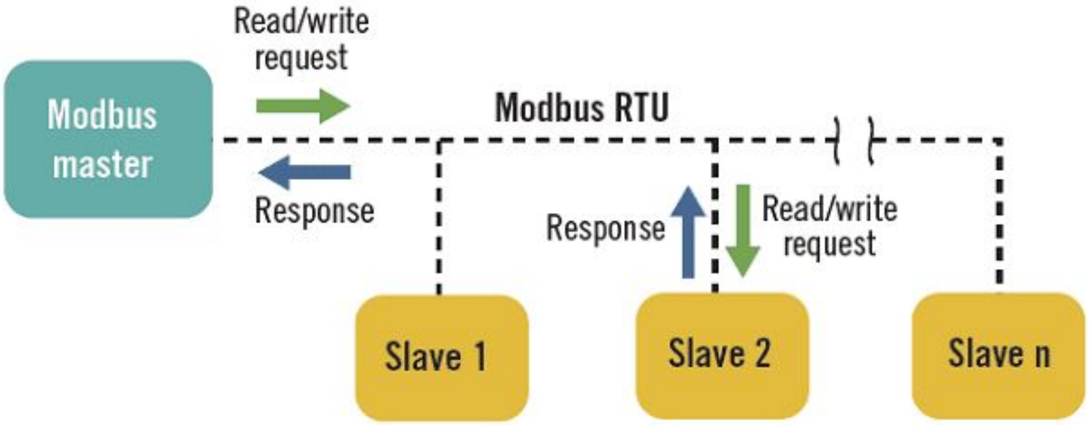
# Philosophy of Modbus

Every message in Modbus deals with reading or writing one of only these four object types:

- Coils
- Discrete Inputs
- Input Registers
- Holding Registers

Single "Controller" that makes requests to "Peripheral" device responses.

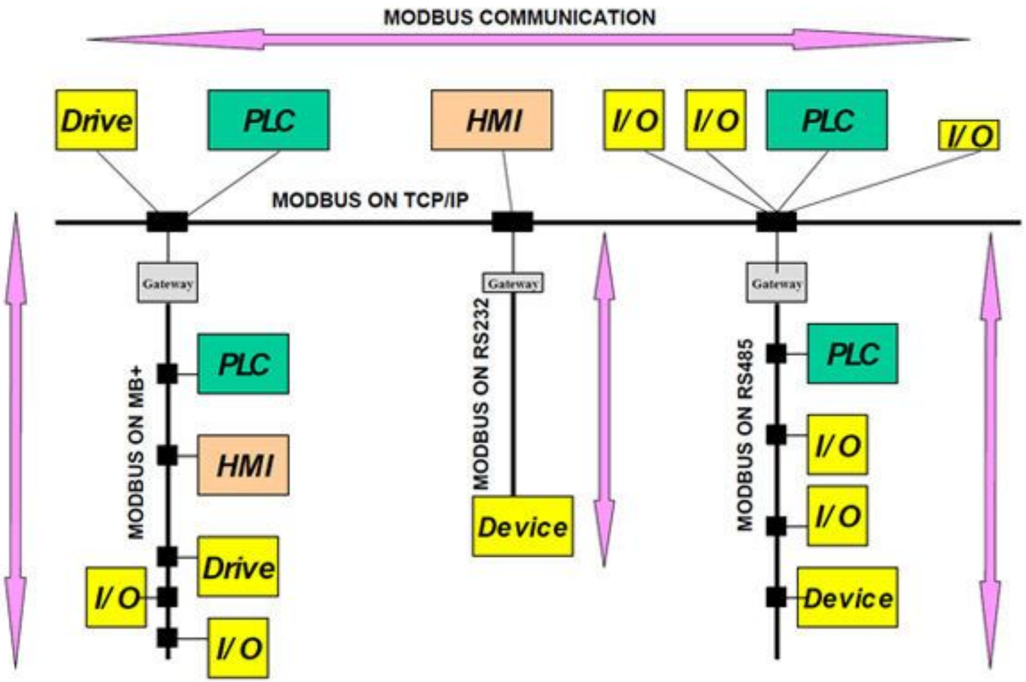
| Object type | Access | Size | Address Space |
|---|---|---|---|
| Coil | Read-write | 1 bit | 00001 – 09999 |
| Discrete input | Read-only | 1 bit | 10001 – 19999 |
| Input register | Read-only | 16 bits | 30001 – 39999 |
| Holding register | Read-write | 16 bits | 40001 – 49999 |



# Modbus Protocols

There are four different protocols that are all called Modbus:

- Modbus RTU
- Modbus ASCII
- Modbus TCP
- Modbus Plus*

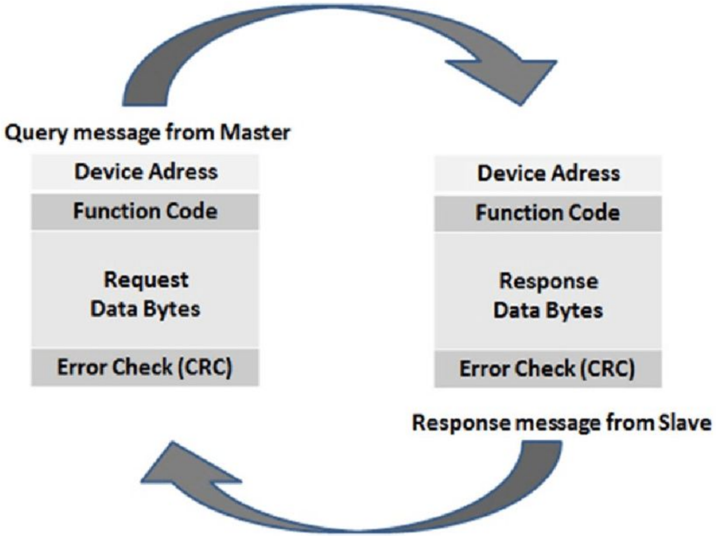


# Mb RTU Message Format

| | | Protocol Data Unit (PDU) | | | |
|---|---|---|---|---|---|
| Start | Slave ID | Function code | Data | CRC error check | Stop |
| 3.5 Bytes | 1 Byte | 1 Byte | n Bytes | 2 Bytes | 3.5 Bytes |
| | Application Data Unit (ADU) | | | | |

**Modbus RTU Frame**

1. There are 3.5 Bytes of of quiet before or after a message is sent
2. An ID of the slave device is next (1 byte)
3. Then a Function Code (1 byte)
4. Then a variable amount of Data
5. Lastly, a CRC error check (2 bytes)




PDU = FC + Data
ADU = Entire message

# Function Codes

Each message sent from the Master will use one and only one of these codes to either read or manipulate a particular value in the Slave's memory tables.

If the slave needs to send back an error code, they will change the first (MSB) of the FC to "1" during the reply.

| Function Code | Action | Table Name |
|---|---|---|
| 01 (01 hex) | Read | Discrete Output Coils |
| 05 (05 hex) | Write single | Discrete Output Coil |
| 15 (0F hex) | Write multiple | Discrete Output Coils |
| 02 (02 hex) | Read | Discrete Input Contacts |
| 04 (04 hex) | Read | Analog Input Registers |
| 03 (03 hex) | Read | Analog Output Holding Registers |
| 06 (06 hex) | Write single | Analog Output Holding Register |
| 16 (10 hex) | Write multiple | Analog Output Holding Registers |

https://www.youtube.com/watch?v=JBGaInI-TG4

# Data Format

The length and content of the data portion will depend on the type of message (FC) being sent.

It will typically start with the address within the appropriate address block then count of how many values are to be read or written to.

| Coil/Register Numbers | Data Addresses | Type | Table Name |
|---|---|---|---|
| 1-9999 | 0000 to 270E | Read-Write | Discrete Output Coils |
| 10001-19999 | 0000 to 270E | Read-Only | Discrete Input Contacts |
| 30001-39999 | 0000 to 270E | Read-Only | Analog Input Registers |
| 40001-49999 | 0000 to 270E | Read-Write | Analog Output Holding Registers |


FC 0x01 = Read Coils

**Request**

| | | |
|---|---|---|
| Function code | 1 Byte | **0x01** |
| Starting Address | 2 Bytes | 0x0000 to 0xFFFF |
| Quantity of coils | 2 Bytes | 1 to 2000 (0x7D0) |

**Response**

| | | |
|---|---|---|
| Function code | 1 Byte | **0x01** |
| Byte count | 1 Byte | **N*** |
| Coil Status | **n** Byte | n = N or N+1 |

Here is an example of a request to read discrete outputs 20–38:

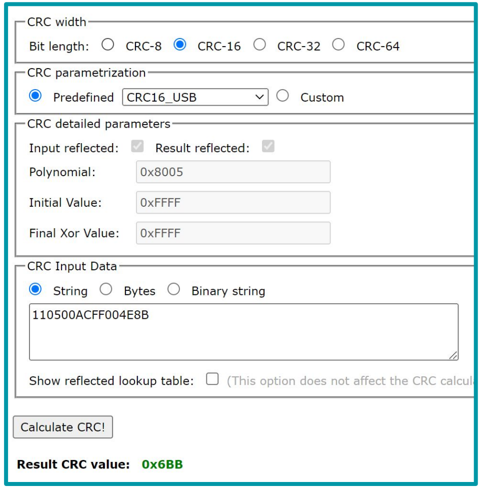
| **Request** | | **Response** | |
|---|---|---|---|
| *Field Name* | *(Hex)* | *Field Name* | *(Hex)* |
| Function | **01** | Function | **01** |
| Starting Address Hi | **00** | Byte Count | **03** |
| Starting Address Lo | **13** | Outputs status 27-20 | **CD** |
| Quantity of Outputs Hi | **00** | Outputs status 35-28 | **6B** |
| Quantity of Outputs Lo | **13** | Outputs status 38-36 | **05** |

# Cyclic Redundancy Check (CRC)

A cyclic redundancy check is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to digital data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption. CRCs can be used for error correction.

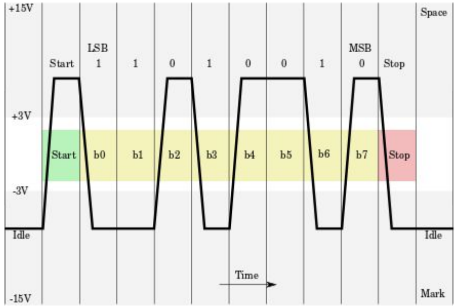
| | | | |
|---|---|---|---|
| CRC-16-IBM | Bisync, Modbus, USB, ANSI X3.28, SIA DC-07, many others; also known as *CRC-16* and *CRC-16-ANSI* | 0x8005 | $x^{16} + x^{15} + x^2 + 1$ |

http://www.sunshine2k.de/coding/javascript/crc/crc_js.html

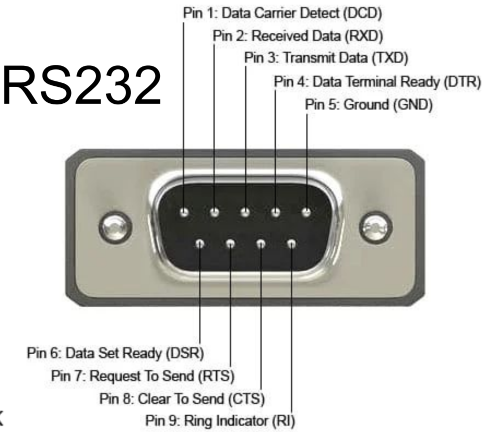


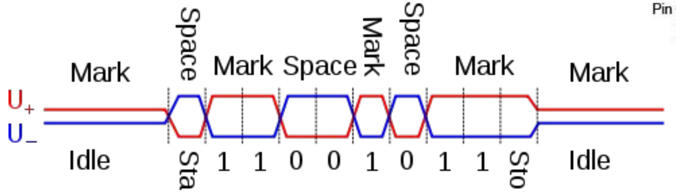
# Physical Layer

- RS-232
  - Three wires (Tx, Rx, Gnd)
  - +V for "0", -V for "1"
- RS-485
  - Two wires (A, B)
  - Differential Voltage
    - A>B for "0", B>A for "1"
- RS-422
  - Four wires (twisted pairs) 2x Tx + Rx
  - Differential Voltage (0-5V)




RS232

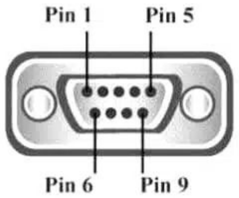



RS422/485

RS422/485 Pinout (9 Pin)

| | |
|---|---|
| Pin 1 | TXD- |
| Pin 2 | TXD+ |
| Pin 3 | RTS- |
| Pin 4 | RTS+ |
| Pin 5 | GND |
| Pin 6 | RXD- |
| Pin 7 | RXD+ |
| Pin 8 | CTS |
| Pin 9 | CTS+ |




https://www.optcore.net/difference-between-rs-232-rs-422-and-rs-485/

# Example Messages

- `11 05 00AC FF00 4E8B`
- `06 0F 0013 000A 02 CD01 BF0B`
- `13 01 0013 0025 0E84` (13 01 05 CD6BB20E1B 45E6)
- `03 06 0001 0003 9A9B`

# AB Micro820 Communications

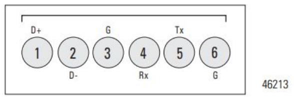
Micro820 controllers support communication through the embedded RS232/RS485 serial port as well as any installed serial port plug-in modules. In addition, Micro820 controllers also support communication through the embedded Ethernet port, and can be connected to a local area network for various devices providing 10 Mbps/100 Mbps transfer rate.

These are the communication protocols supported by Micro820 controllers:

- Modbus RTU Master and Slave
- CIP Serial Client/Server (RS232 only)
- ASCII
- EtherNet/IP Client/Server
- Modbus TCP Client/Server
- CIP Symbolic Client/Server
- DHCP Client
- Sockets Client/Server TCP/UDP

https://literature.rockwellautomation.com/idc/groups/literature/documents/um/2080-um005_-en-e.pdf

*Serial Port Terminal Block*

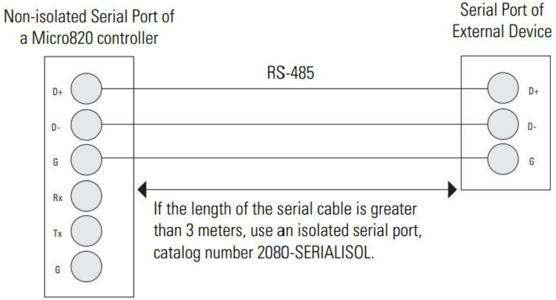


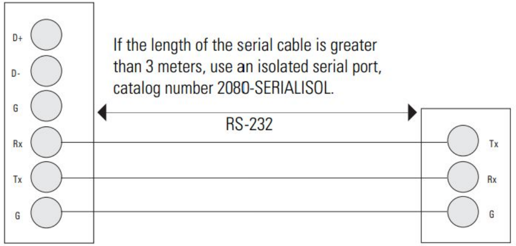

(View into terminal block)

| | | |
|---|---|---|
| Pin 1 | RS485 | Data + |
| Pin 2 | RS485 | Data - |
| Pin 3 | RS485 | Ground[1] |
| Pin 4 | RS232 | Receive |
| Pin 5 | RS232 | Transmit |
| Pin 6 | RS232 | Ground[1] |

(1) Non-isolated.

**Serial Port Wiring**



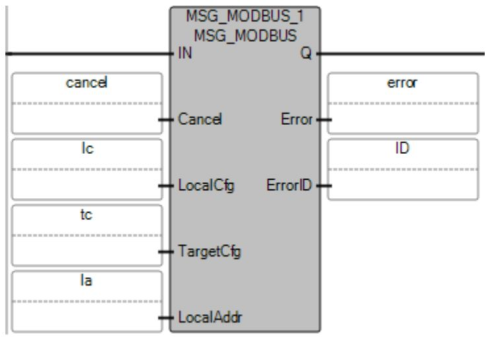

**IMPORTANT** Do not connect G terminals of the serial port to Earth/Chassis ground.

# Modbus RTU on AB micro820

| Parameter | Parameter Type | Data Type | Description |
|---|---|---|---|
| IN | Input | BOOL | Rung input state.<br>TRUE - Rising Edge detected, start the instruction block with the precondition that the last operation has been completed.<br>FALSE - Rising Edge not detected, not started. |
| Cancel | Input | BOOL | TRUE - Cancel the execution of the instruction block.<br>FALSE - when IN is TRUE.<br>Cancel input is dominant. |
| LocalCfg | Input | MODBUSLOCPARA | Define structure input (local device).<br>Define the input structure for the local device using the MODBUSLOCPARA data type on page 179. |
| TargetCfg | Input | MODBUSTARPARA | Define structure input (target device).<br>Define the input structure for the target device using the MODBUSTARPARA data type on page 182. |
| LocalAddr | Input | MODBUSLOCADDR | MODBUSLOCADDR is a 125 Word array that is used by Read commands to store the data (1-125 words) returned by the Modbus slave and by Write commands to buffer the data (1-125 words) to be sent to the Modbus slave. |
| Q | Output | BOOL | Outputs of this instruction are updated asynchronously from the program scan. Output Q cannot be used to re-trigger the instruction since IN is edge triggered.<br>TRUE - MSG instruction finished successfully.<br>FALSE - MSG instruction is not finished. |
| Error | Output | BOOL | Indicates an error occurred.<br>TRUE - An error is detected.<br>FALSE - No error. |
| ErrorID | Output | UINT | A unique numeric that identifies the error. The errors for this instruction are defined in MSG_MODBUS error codes. |

| Parameter | Data type | Description |
|---|---|---|
| Channel | UINT | Micro800 PLC serial port number:<br>• 2 for the embedded serial port, or<br>• 5-9 for serial port plug-ins installed in slots 1 through<br>• 5 for slot 1<br>• 6 for slot 2<br>• 7 for slot 3<br>• 8 for slot 4<br>• 9 for slot 5 |
| TriggerType | USINT | Represents one of the following:<br>• 0: Msg Triggered Once (when IN goes from False to True)<br>• 1: Msg triggered continuously when IN is True<br>• Other value: Reserved |
| Cmd | USINT | Represents one of the following:<br>• 01: Read Coil Status (0xxxx)<br>• 02: Read Input Status (1xxxx)<br>• 03: Read Holding Registers (4xxxx)<br>• 04: Read Input Registers (3xxxx)<br>• 05: Write Single Coil (0xxxx)<br>• 06: Write Single Register (4xxxx)<br>• 15: Write Multiple Coils (0xxxx)<br>• 16: Write Multiple Registers (4xxxx) |

**MSG_MODBUS Ladder Diagram example**





https://literature.rockwellautomation.com/idc/groups/literature/documents/rm/2080-rm001_-en-e.pdf

https://www.youtube.com/watch?v=ARg2QHn3IB0

## Mapping Address Space and supported Data Types

Since Micro800 uses symbolic variable names instead of physical memory addresses, a mapping from symbolic Variable name to physical Modbus addressing is supported in Connected Components Workbench software, for example, InputSensorA is mapped to Modbus address 100001.

By default Micro800 follows the six-digit addressing specified in the latest Modbus specification. For convenience, conceptually the Modbus address is mapped with the following address ranges. The Connected Components Workbench mapping screen follows this convention.

| Variable Data Type | 0 - Coils 000001 to 065536 | | 1 - Discrete Inputs 100001 to 165536 | | 3 - Input Registers 300001 to 365536 | | 4 - Holding Registers 400001 to 465536 | |
|---|---|---|---|---|---|---|---|---|
| | Supported | Modbus Address Used | Supported | Modbus Address Used | Supported | Modbus Address Used | Supported | Modbus Address Used |
| BOOL | Y | 1 | Y | 1 | | | | |
| SINT | Y | 8 | Y | 8 | | | | |
| BYTE | Y | 8 | Y | 8 | | | | |
| USINT | Y | 8 | Y | 8 | | | | |
| INT | Y | 16 | Y | 16 | Y | 1 | Y | 1 |
| UINT | Y | 16 | Y | 16 | Y | 1 | Y | 1 |