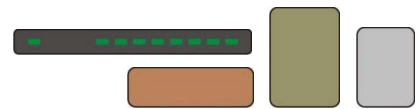




# The Modbus Driver

---



The Modbus driver allows North to interface with a wide range of equipment supporting the Modbus protocol, both over TCP/IP and serial connections. As a client, the driver can request values from Modbus devices and, as a server, provide values from Essential Data to Modbus clients. Available for Commander and ObSys.

This document relates to Modbus driver version 3.1

Please read the *Commander Manual* or *ObSys Manual* alongside this document, available from [www.northbt.com](http://www.northbt.com)

# Contents

Compatibility with the Modbus System.....	3
Equipment .....	3
Values .....	4
Prerequisites .....	5
Using the Driver .....	6
Starting the Interface.....	6
Making the Cable .....	6
Setting up the Driver.....	6
Checking Communications .....	7
Operation as a Modbus Client .....	8
Data Model .....	8
Supported Function Codes.....	8
Value Decoding .....	9
Operation as a Modbus Server .....	13
Essential Data Value Translation.....	13
Supported Function Codes.....	14
Exporting the Modbus Register List .....	14
Object Specifications.....	15
Example Object Reference .....	15
Device Top-Level Objects .....	15
Modbus Setup .....	16
Modbus TCP Client Setup .....	17
TCP Unit .....	17
Modbus TCP Server Setup .....	18
Network.....	19
Network Interface .....	19
Modbus Serial Setup.....	20
Modbus Serial Client Setup .....	21
Serial Unit.....	21
Modbus Serial Server Setup .....	22
Security .....	23
Register List.....	24
Register .....	24
Advanced Settings .....	25
Formula Setup .....	26
Modbus System.....	27
Default Modbus Device .....	28
Appendix A: Modbus Integration Summary .....	34
Modbus over TCP/IP .....	34
Modbus over Serial-line.....	34
Function Codes .....	34
Register Address .....	34
Register Value .....	35
Checking Values .....	35
Driver Versions .....	36

# Compatibility with the Modbus System

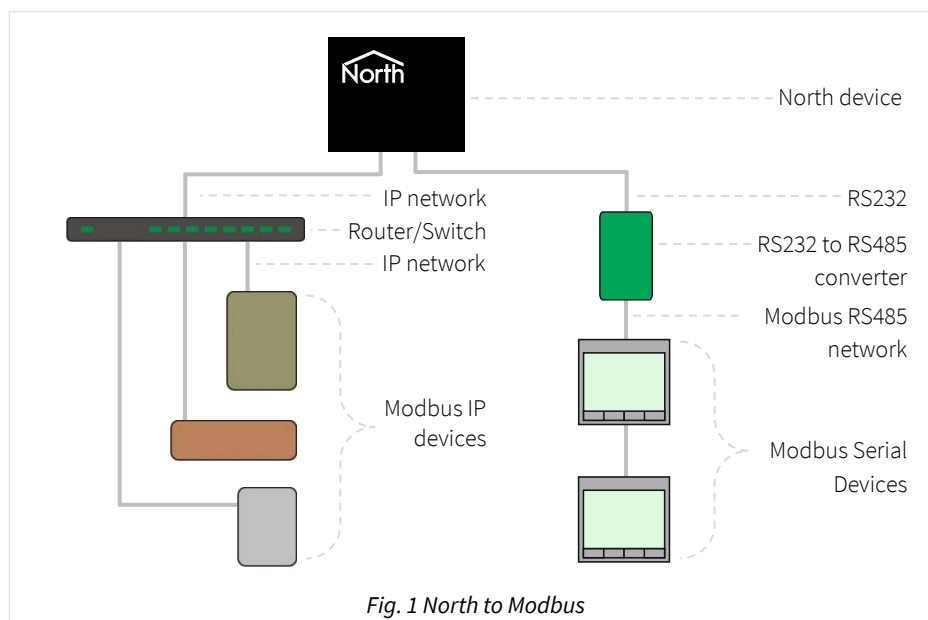
The Modbus driver allows North to interface with a wide range of equipment supporting the Modbus over TCP/IP, Modbus over serial-line, and JBus protocols. The driver can both request values from Modbus devices, and provide values to a Modbus front-end when requested.

Modbus uses a client-server model. As a client, the driver is capable of both requesting values from Modbus units (i.e. power meters, controllers, etc.), and as a server, providing values from the Essential Data and Extra Data within the North device when requested by a Modbus client (i.e. display or front-end).

The driver connects to an IP network, and can access up to 30 Modbus TCP/IP units on the network (Fig. 1). The driver is also capable of being accessed by two TCP/IP client devices simultaneously.

The driver also connects using the North device's serial port, typically via an RS485 converter, to Modbus over serial-line equipment (Fig. 1). In client mode, a network of Modbus units can be accessed (see [Prerequisites: Driver as Modbus Serial Client](#) for limitations). In server mode, the driver is capable of being accessed by a single client/master device.

The JBus protocol is fully compatible with Modbus over serial-line.



## Equipment

Many different types of Modbus equipment are compatible with the driver, including:

- Energy meters
- Generators
- Variable speed drives
- PLCs
- AHUs
- Control systems

Equipment is available from many different manufacturers, including ABB, Autometers, Carel, Carlo Gavazzi, Carrier, Ciat, Daikin, Danfoss, Mitsubishi, Schneider, Siemens, Socomec, Stulz, Swegon, Tyco, plus many more.

*Application Notes* are available for some of these devices, search North product documentation.

## Values

The Modbus driver can act as a Modbus client, reading and writing values from a Modbus device elsewhere on the Modbus network. The driver can also act as a Modbus server, allowing other Modbus clients to read and write values within the Essential Data.

### Driver as Modbus Client

Depending on the type of Modbus equipment, each Modbus device can have the following primary value types available:

- **Coils** – digital output, e.g. enable command
- **Discrete Inputs** – digital input, e.g. off/on, or alarm states
- **Holding registers** – analogue output, e.g. setpoint values
- **Input registers** – analogue input, e.g. meter readings

Read the Modbus register list, available from the equipment manufacturer, for the values available from a specific device.

### Driver as Modbus Server

The driver presents values from the Essential Data and Extra Data as Modbus values, accessible to any client device on the Modbus network. Essential Data contains 640 values on Commander, and 1280 values on ObSys. If necessary, start the Extra Data driver (which requires an interface licence) for an additional 1024 values. Access to these values can be controlled by configuring privilege levels within the driver.

## Prerequisites

If an *Application Note* is not available for your Modbus unit, you will need a Modbus register list from the equipment manufacturer. This should describe the function codes or commands supported, register addresses available, and how register values are stored (16-bit integer, 32-bit integer, IEEE float, multipliers, etc.)

### Driver as Modbus TCP Client

All Modbus devices must be configured with a unique IP address on the TCP/IP network. If the TCP port can be configured, then this should typically be set to 502. If you are connecting to Modbus over serial devices via a gateway, each device must be assigned a unique serial address.

If you are connecting to the Modbus IP network via a firewall, then the driver will require outbound access to controllers on TCP port 502 (Port 502 is reserved by IANA for use by Modbus).

### Driver as Modbus TCP Server

The driver will respond to requests from a Modbus client with a unit identifier of 255, 0, or 1.

### Driver as Modbus Serial Client

We recommend installing only Modbus devices of the same type (manufacturer and model) on a network. Different device types may be incompatible for the following reasons – baud rate, byte format, inter-frame delay, and RS485 isolation.

All Modbus devices must be configured with a unique address on the network, and the following common parameters: baud rate, byte format.

Modbus RTU operating mode is supported.

An RS232-485 adapter is required for RS485 devices. Set the baud rate and data bits to match the devices.

The RS485 standard allows at least 32 devices on a network. However, the maximum number depends on the unit load of each device on the network – typically 64 devices with a 0.5 unit load, or 128 devices with a 0.25 unit load. The Modbus standard and this driver support up to 247 addresses.

### Driver as Modbus Serial Server

Configure the driver with a unique address on the Modbus network, and the following common parameters: baud rate, byte format.

Modbus RTU operating mode is supported.

# Using the Driver

On ObSys and Commander, the Modbus driver is pre-installed. Using all of these North devices, you can use the driver to create an interface to a Modbus system. Once started, you will need to set up the driver before it can communicate with the Modbus system.

The Modbus driver uses zero licence units.

## Starting the Interface

- 🖥 To start an interface using the Modbus driver, follow these steps:
  - **Start Engineering** your North device using ObSys
  - Navigate to **Configuration, Interfaces**, and set an unused **Interface** to 'Modbus' to start the particular interface
  - Navigate to the top-level of your North device and re-scan it

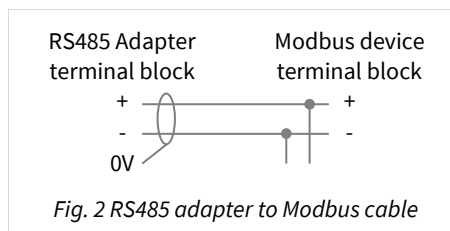
The driver setup object (Mc), labelled **Modbus Setup**, should now be available.

## Making the Cable

### RS485 Devices

Connect the North device COM port to an RS232 to RS485 adapter.

Using the RS485 cable specification (Fig. 2), connect the RS485 adapter to the Modbus device network.



RS485 adapters are available from North, order code MISC/RS232/485. This adaptor has a unit load of 1.

### RS232 Devices

Connect the North device COM port to the Modbus device.

The cable specification will change for each type of Modbus device. Read the *Application Note* or manufacturer's documentation on your specific model for more information.

The maximum RS232 cable length is 15m and should be as short possible.

## Setting up the Driver

- 🖥 To set up the driver, follow these steps:
  - Navigate to the **Modbus Setup** object (Mc). For example, if you started interface 1 with the driver earlier, then the object reference will be 'M1'
  - Navigate to **Modbus TCP Client Setup, TCP Unit 1** and set the **Label** and **IP Address** of a Modbus TCP unit that you wish to access values within. If the device is connected via a Modbus gateway, then set the **Serial Address**
  - Follow additional steps on the *Application Note*, if available for the device
  - Repeat for each Modbus TCP unit that your wish to access

- Navigate to **Modbus Serial Setup** and set the **RS232 Com Port** to the port number of the North device you are connecting to Modbus
- Set **Modbus Serial Mode** to 'Client' or 'Server' operation
- Set **Baud Rate** to match the Modbus devices, typically 9600, 19200 or 38400 baud
- Set the **Byte Format** to the parity and stop bits configured in the Modbus devices
- If 'Client' mode is enabled, navigate to **Serial Client Setup** and follow additional steps on the *Application Note*, if available for the device
- If 'Server' mode is enabled, navigate to **Serial Server Setup** and set the **Address on Modbus network** to a unique address on the network.

## Checking Communications

To check Modbus client operation, scanning the **Modbus System** will first respond with all units configured with an IP address in Modbus TCP Client Setup, and then automatically detect any Modbus serial units connected. You can check a unit is communicating by and viewing values within it.

To check Modbus server operation, navigate to **Modbus TCP Server Setup** then **Network** to view interfaces open on the North device and their IP address.

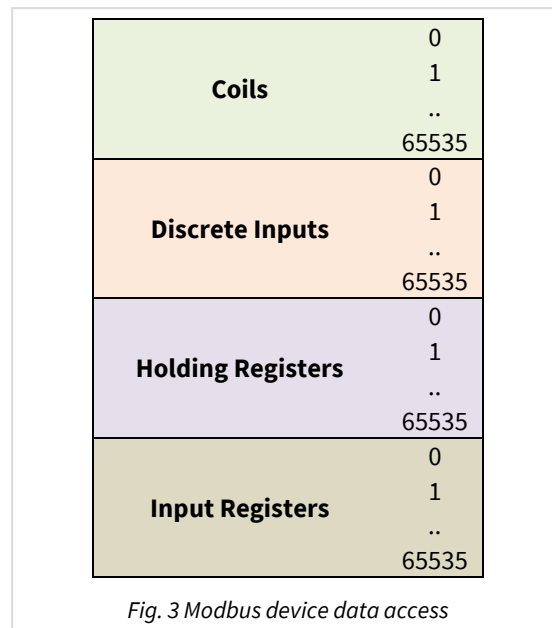
# Operation as a Modbus Client

## Data Model

A Modbus server device stores data using four primary tables, which can each be accessed by the driver.

Table	Data type	Adjustable	Used for
Coils	1-bit data values	Read-Write	Digital outputs
Discrete Inputs	1-bit data values	Read-only	Digital inputs
Holding Registers	16-bit data values	Read-Write	Analogue outputs: setpoints, calculated values
Input Registers	16-bit data values	Read-only	Analogue inputs: sensor readings, meter values

Each table can contain up to 65536 entries, addressed in the range 0 to 65535 (Fig. 3).



Implementations of the Modbus protocol in a device can vary:

- It is common that these four tables may overlap in a device, so a single address range is used
- Distinctions between inputs and outputs, or 1-bit and 16-bit data values may be blurred
- Multiple consecutive table entries are combined to store a larger data value, e.g. 32-bit or 64-bit data values
- Register addresses are often documented in the range 1 to 65536. To rescale these in the 0 to 65535 range of the Modbus protocol, you will need to subtract 1. E.g. Register 100 becomes 99.

## Supported Function Codes

A client can read and write values in the tables using different Modbus function codes. A particular server device may only support some of these codes. The driver function is used as part of the object reference described later.

Function Code (read)	Function Code (write)	Action	Driver Function
01	05	Read/Write Coil	C
01	15	Read/Write Coil	U
02		Read Discrete Input	A
03	06	Read/Write single Holding Register	D
03	16	Read/Write multiple Holding Registers	E to L
04		Read single Input Register	B
04		Read multiple Input Registers	M to T



## Value Decoding

The Modbus protocol only describes storing values as either a 1-bit digital or 16-bit register value. All implementations of Modbus have variations from this, including:

- 32-bit and 64-bit integer values (including LSW-MSW order)
- IEEE floating point values (including LSW-MSW order)
- Bit array in register
- Multipliers to change register data from integer
- ASCII string
- Byte-order changed

The driver has several decode types available that are used to translate a raw Modbus register, from a controller, into a value. The decode is used as part of the object reference described later.

### Digital State (Decode A)

The value stored within a discrete input or coil entry is always 0 or 1, and always decodes to 0 or 1.

### Unsigned Integer (Decode B and O)

The value stored within the register entry decodes to an unsigned number. For a single register, this will be in the range 0 to 65535.

Either one, two, or four registers can be decoded to a 16-bit, 32-bit, or 64-bit value respectively. In multi-register values, decode B assumes MSW in the first register, and decode O assumes LSW in the first register.

#### Examples

The single register value of 0x4849 (hex) will decode to 18505 (decimal). The multi-register values of 0x0012 and 0x3456 will decode to 1193046.

### Signed Integer (Decode C and P)

The value stored within a register entry decodes to a signed integer number. For a single register, this will be in the range -32768 to 32767.

Either one, two, or four registers can be decoded to a 16-bit, 32-bit, or 64-bit value respectively. In multi-register values, decode C assumes MSW in the first register, and decode P assumes LSW in the first register

#### Examples

The single register value of 0x4849 (hex) decodes to 18505 (decimal), and the single register value of 0xB7B7 decodes to -18505.

### BCD in Lower Nibbles (Decode D)

The value stored within the lower nibbles of the register(s) decodes to a binary decoded decimal value.

One, two, three, or four register values can be decoded.

#### Example

The two lower nibbles of a single register decodes as  $\text{Sum}(V): 80+8+1 = 89$

Bit value (V)	Unused nibble				Lower nibble				Unused nibble				Lower nibble			
					80	40	20	10					8	4	2	1
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1

## BCD in Register (Decode E)

The value stored within the register(s) decodes to a binary coded decimal value.

One, two, three, or four register values can be decoded.

### Example

A single register value of 0x4849 (hex) decodes as Sum(V):  $4000+800+40+8+1 = 4849$

Bit value (V)	8000	4000	2000	1000	800	400	200	100	80	40	20	10	8	4	2	1
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1

## ASCII value in LSB (Decode F)

The value stored within the least significant byte of the register(s) decodes to a single ASCII character. Up to 16 registers can be accessed at once, i.e. 16 characters.

### Example

The LSB of a single register decodes as a single character:  $64+8+1 =$  ASCII code 73 = 'I'

ASCII character																
Bit value	128 64 32 16 8 4 2 1															
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1

## ASCII String (Decode G)

The value stored within each register decodes to two ASCII characters. Up to 16 registers can be accessed at once, i.e. 32 characters.

### Example

A single register decodes as two characters:  $64+8 =$  ASCII code 72, and  $64+8+1 =$  ASCII code 73. The full string is 'HI'.

	First ASCII character								Second ASCII character							
Bit value	128	64	32	16	8	4	2	1	128	64	32	16	8	4	2	1
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1

## Unsigned Integer in LSB (Decode H)

The value stored within the least significant byte (LSB) of a single register decodes to an unsigned number, a byte in the range 0 to 255. When writing, the register is first read to preserve the MSB.

### Example

The LSB of a single register decodes as Sum(V):  $64+8+1 = 73$

	Unused MSB								LSB							
Bit value (V)									128	64	32	16	8	4	2	1
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1

## Unsigned Integer in MSB (Decode I)

The value stored within the most significant byte (MSB) of a single register decodes to an unsigned number, a byte in the range 0 to 255. When writing, the register is first read to preserve the LSB.

### Example

The MSB of a single register decodes using Sum(V):  $64+8 = 72$

	MSB								Unused LSB							
Bit value (V)	128	64	32	16	8	4	2	1								
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	

## IEEE Float (Decode J and M)

The value stored within two registers decodes to an IEEE floating-point number. A four register value can be decoded to a double-precision floating-point number.

Use decode type J for big-endian values (MSW in first register and LSW in second), or decode type M for little-endian values (LSW in first register and MSW in second).

Decode type J can also decode a single register as a half-precision floating-point number.

## Single Bit of Register (Decode K)

Returns the specified bit from a register value. Bits are indexed starting with the most significant byte (MSB) of the first register. This is shown below as bit index 7 to 0 (MSB), followed by 15 to 8 (LSB). Note how this differs from the traditional bit numbering of a 16-bit value (shown 15 to 0 below the register value).

	MSB								LSB							
Bit index (K)	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
Register value																
Bit number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Multiple register values are supported. When writing, the register is first read to preserve remaining bits.

### Example

If a register has the value 0x4849 (hex). Bit indexes 8, 11, 14, 3, and 6 have the value 1. All other bits have the value 0.

Bit index	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1

## Bit Mask (Decode L)

Performs a bitwise AND operation on a register value and mask value. The result indicates which bits of the mask value are also set in the register value.

### Example

If a register has the value 18505 (decimal). To find the value of bits 3 to 6, we first calculate the mask value:  $\text{Sum}(V) = 64 + 32 + 16 + 8 = 120$ . So,  $\text{Register AND Mask} = 18505 \text{ AND } 120 = 72$ .

Bit values (V)	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1
Mask value	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0

## Bit Mask with Bit Shift (Decode N)

Performs a bitwise AND operation on a register value and mask value, followed by a bit-shifting operation to move all bits to the right by the specified number.

### Example

If a register holds a value in bits 3 to 6. First extract this value using the AND Mask value 120 ( $64 + 32 + 16 + 8 = 120$ ). Then bit shift this value 3 positions to the right (rescaling the value to base 0).

If a register has the value 18505 (decimal):  $18505 \text{ AND } 120 = 72 \gg 3 = 9$

Bit values	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Mask value	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
Register value	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1
Bit shift result	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

## Single Register of Multi-Register (Decode R)

Returns the specified register index of a multi-register value.

The value stored within the single register entry decodes to an unsigned number, in the range 0 to 65535.

The multi-register value is temporarily cached by the driver, so that on decoding a different register index it uses the same multi-register value. The life of the cache can be adjusted using the Cache Life driver object (A.TC).

### Examples

The two register value of 0x0012 and 0x3456, will decode to 0x0012 when requesting register index 1, and 0x3456 when requesting register index 2.

# Operation as a Modbus Server

The values from Essential Data may be accessed using any of the supported Modbus function codes. A single address range is used for all functions – 0...639 on Commander, and 0...1279 on ObSys.

Both the Modbus TCP Server Setup and Serial Server Setup objects contain a **Register List** (RL) object detailing the registers available from the North device. [See below](#) on how to export this Modbus register address mapping list for a third-party.

If ExtraData is used, the extra 1024 values appear in registers that follow on from the existing Essential Value registers – i.e. Registers 640...1663 on Commander, or 1280...2303 on ObSys.

## Essential Data Value Translation

The North device has only one data table – Essential Data – and so maps all Modbus table requests to Essential Data. The Essential Data value is translated to a Modbus register or state depending which Modbus Table is specified and how the value is configured in Essential Data:

Modbus Table	Number	Essential Data Type		
		Float	NoYes or OffOn	Enum
Input Register Holding Register	16-bit unsigned register value in the range 0...65535.	16-bit signed value in the range -32768...32767	‘No’ and ‘Off’ states are converted to the value 0	16-bit unsigned register value in the range 0...65535
		Value scaled $10^{dp}$ ( $dp$ is the number of decimal places configured in Essential Data)	‘Yes’ and ‘On’ states are converted to the value 1	
			16-bit unsigned register value in the range 0...1.	
Discrete Input Coil	Binary on/off state. If the value is zero (0) then an ‘off’ state is returned, any other value returns an ‘on’ state			

When reading, if the Essential Data object has an Access Security level that does not allow Modbus access to it (see [Security](#)), a '0' value is returned. When writing, single value writes (Function Codes 05 and 06) will have an error message returned; multi-value writes (Function Codes 15 and 16) will not.

If the Essential Data object has Adjustable set to 'No', and the incoming message attempts to write to it, the write does not occur: single value writes (Function Codes 05 and 06) will be returned an error message, multi-value writes (Function Codes 15 and 16) will not.


## Supported Function Codes

The driver supports the following Function Codes. Some Modbus messages can contain a quantity of items, and the table shows the range of quantities supported by the driver for the various codes:

Function Code	Action	Quantity
01	Read Coils	1 to 255
02	Read Discrete Inputs	1 to 255
03	Read Holding Registers	1 to 125
04	Read Input Registers	1 to 125
05	Write Single Coil	n/a
06	Write Single Holding Register	n/a
15	Write Multiple Coils	1 to 255
16	Write Multiple Holding Registers	1 to 123

## Exporting the Modbus Register List

Both the Modbus TCP Server Setup and Serial Server Setup objects contain the same **Register List** (RL) object detailing the registers available from the North device.

-  To export the North device's Modbus register list, follow these steps:
- Navigate to the **Modbus Setup** object (Mc). For example, if you started interface 1 with the driver earlier, then the object reference will be 'M1'
  - Navigate to **Modbus TCP Server Setup, Register List**
  - From the menu, select **Extras, Export to CSV...**

# Object Specifications

Once an interface is started, one or more extra objects become available within the top-level object of the device. As with all North objects, each of these extra objects may contain sub-objects, (and each of these may contain sub-objects, and so on) – the whole object structure being a multi-layer hierarchy. It is possible to navigate around the objects using the ObSys Engineering Software.

Each object is specified below, along with its sub-objects.

## Example Object Reference

An example of a reference to an object in the same device: the Modbus System (S1) contains a Unit (U1), which contains an Input register (B2.B). Therefore, the object reference will be 'S1.U1.B2.B'.

An example of a reference to an object in a different device: the IP network object (IP) contains Default Commander object (CDIP), which contains the object above (S1.U1.B2.B) – therefore the complete object reference is 'IP.CDIP.S1.U1.B2.B'.

## Device Top-Level Objects

When an interface is started using the Modbus driver, the objects below become available within the top-level object of the device. For example, if Interface 1 is started, then the object with references 'M1' and 'S1' become available.

Description	Reference	Type
<b>Modbus Setup</b> Set up the Modbus driver, started on interface c (c is the interface number)	Mc	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31]</i>
<b>Modbus System</b> Access Modbus systems connected to interface c (c is the interface number)	Sc	Variable Container: <i>[Modbus]</i>

# Modbus Setup

Object Type: *[OSM v20\Modbus v31]*

Object Type: *[CDM v20\Modbus v31]*

Object Type: *[OSM v20\Modbus v30]*

Object Type: *[CDM v20\Modbus v30]*

The Modbus Setup Module contains the following objects:

Description	Reference	Type
<b>System Label</b> Label displayed when scanning the system	DL	Obj\Text: 20 chars max; Adjustable
<b>Modbus TCP Client Setup</b> Enable and configure Modbus TCP client operation	TC	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Client]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Client]</i>
<b>Modbus TCP Server Setup</b> Enable and configure Modbus TCP server operation	TS	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Server]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Server]</i>
<b>Modbus Serial Setup</b> Enable and configure Modbus serial client or server operation	RS	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Serial]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Serial]</i>
<b>Advanced Settings</b> Set advanced parameters for Modbus operation	A	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Advanced]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Advanced]</i>



# Modbus TCP Client Setup

Object Type: [OSM v20\Modbus v31\Client]

Object Type: [CDM v20\Modbus v31\Client]

Object Type: [OSM v20\Modbus v30\Client]

Object Type: [CDM v20\Modbus v30\Client]

The Modbus TCP Client Setup object is used to enable client operation in the driver, and add the details of Modbus TCP devices available on the IP network.

Description	Reference	Type
<b>Enable TCP Client</b>	E	Obj\NoYes; Adjustable
<b>TCP Unit x</b> Configure Modbus TCP client device address. Unit number, x, is in the range 1...30	Ux	Fixed Container: On the Commander platform this will be [CDM v20\Modbus v31\Client\Unit] On the ObSys platform this will be [OSM v20\Modbus v31\Client\Unit]

## TCP Unit

Object Type: [OSM v20\Modbus v31\Client\Unit]

Object Type: [CDM v20\Modbus v31\Client\Unit]

Object Type: [OSM v20\Modbus v30\Client\Unit]

Object Type: [CDM v20\Modbus v30\Client\Unit]

The TCP Unit object is used to configure the address of a Modbus TCP device on the network.

Description	Reference	Type
<b>Label</b> Label displayed when scanning the Modbus system	L	Obj\Text: 20 chars; Adjustable
<b>IP Address</b>	IA	Obj\IP; Adjustable
<b>TCP Port</b>	PN	Obj\Num: 1...65535; Adjustable Default: 502
<b>Serial Address</b> If connecting to a single Modbus TCP device, set to the address 255. If connecting via a Modbus gateway, set to the Modbus over serial device address.	A	Obj\Num: 1...255; Adjustable Value 255: Modbus TCP device (default) Value 1...247: Modbus serial address
<b>Device Type</b> Leave blank to view default Modbus objects for a device. Refer to <i>Application Note</i> for other types available	DT	Obj\Text: 20 chars; Adjustable

# Modbus TCP Server Setup

Object Type: *[OSM v20\Modbus v31\Server]*

Object Type: *[CDM v20\Modbus v31\Server]*

Object Type: *[OSM v20\Modbus v30\Server]*

Object Type: *[CDM v20\Modbus v30\Server]*

The Modbus TCP Server Setup object is used to enable server operation in the driver, and select which network interface the protocol is accessible on.

Description	Reference	Type
<b>Enable TCP Server</b>	E	Obj\NoYes; Adjustable
<b>Network</b> Configure the local IP address and TCP port for Modbus	N	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Network]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Network]</i>
<b>Security</b> Configure privilege levels to control read and adjust access to Essential Data and Extra Data from a Modbus TCP client	S	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Privs]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Privs]</i>
<b>Register List</b> List of registers made available from Essential Data to a connected Modbus TCP client. Useful for documentation.	RL	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\RegList]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\RegList]</i>

## Network

Object Type: [OSM v20\Modbus v31\Network]

Object Type: [CDM v20\Modbus v31\Network]

Object Type: [OSM v20\Modbus v30\Network]

Object Type: [CDM v20\Modbus v30\Network]

Configure the Modbus Network connectivity using this object. By default, all available IP addresses are opened for requests on TCP port 502.

If required, change the TCP port number or restrict access to a single IP address on the North device.

Description	Reference	Type
<b>Interfaces open</b> Reports the number of interfaces Modbus is available on	C	Obj\Num: 0...8
<b>Force single IP address</b> Force the driver to use only one of the interface IP addresses. By default, all are used when the address is '0.0.0.0'	IA	Obj\IP; Adjustable
<b>TCP Port</b> TCP port number	PN	Obj\Num: 1...65535; Adjustable Default 502
<b>Interface x</b> Status information for a network interface available on the North device. The interface number, x, is in the range 1...4 on Commander and 1...8 on ObSys	Ix	Fixed Container: On the Commander platform this will be <a href="#">[CDM v20\Modbus v31\Network\Interface]</a> On the ObSys platform this will be <a href="#">[OSM v20\Modbus v31\Network\Interface]</a>

## Network Interface

Object Type: [OSM v20\Modbus v31\Network\Interface]

Object Type: [CDM v20\Modbus v31\Network\Interface]

Object Type: [OSM v20\Modbus v30\Network\Interface]

Object Type: [CDM v20\Modbus v30\Network\Interface]

An Interface represents a physical or virtual network interface on the North device. Use this object to find out the IP address available and if Modbus TCP has opened a port to listen for requests.

Description	Reference	Type
<b>IP address</b> IP address available for the interface	IA	Obj\IP
<b>Port open</b> Indicates if Modbus has opened a port on the interface	S	Obj\NoYes

# Modbus Serial Setup

Object Type: *[OSM v20\Modbus v31\Serial]*

Object Type: *[CDM v20\Modbus v31\Serial]*

Object Type: *[OSM v20\Modbus v30\Serial]*

Object Type: *[CDM v20\Modbus v30\Serial]*

The Modbus Serial Setup object is used to select client or server operation in the driver, and configure the serial port.

Description	Reference	Type
<b>Modbus Serial Mode</b> Select operating mode for driver on the serial interface. Select 'Client' to request values from other Modbus devices. Select 'Server' if a Modbus master/client will request values from the North device	M	Obj\Enum; Adjustable Values: 0=None, 1=Client, 2=Server
<b>RS232 COM Port</b>	COM	Obj\Num: 1...8; Adjustable
<b>Baud Rate</b>	BR	Obj\Num; Adjustable; Default: 19200 Range: 1200, 2400, 4800, 9600, 19200 or 38400
<b>Byte Format</b> Set the parity and stop bits	BF	Obj\Enum: 0...9; Adjustable; Default: 8 (Even/1) See note 1
<b>Serial Client Setup</b> Configure Modbus serial client operation	C	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Serial\Client]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Serial\Client]</i>
<b>Serial Server Setup</b> Configure Modbus serial server operation	S	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Serial\Server]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Serial\Server]</i>

## Notes

- 1 Modbus RTU uses 8 data bits. Byte format can have the following values:

Value	Parity	Stop bits	Notes	Data Format for RS232-485 converter
0	None	1	2 stop bits are recommended when using no parity	10-bits
1	None	2		11-bits
4	Odd	1		11-bits
5	Odd	2		12-bits
8	Even	1	Default value	11-bits
9	Even	2		12-bits

# Modbus Serial Client Setup

Object Type: [OSM v20\Modbus v31\Serial\Client]

Object Type: [CDM v20\Modbus v31\Serial\Client]

Object Type: [OSM v20\Modbus v30\Serial\Client]

Object Type: [CDM v20\Modbus v30\Serial\Client]

The Modbus Serial Client Setup object is used to optionally set a default device type and add details of Modbus serial devices available on the RS485 network.

Description	Reference	Type
<b>Serial Unit x</b> Configure Modbus client device address. Unit number, x, is in the range 1...30	Ux	Fixed Container: On the Commander platform this will be [CDM v20\Modbus v31\Serial\Unit] On the ObSys platform this will be [OSM v20\Modbus v31\Serial\Unit]

## Serial Unit

Object Type: [OSM v20\Modbus v31\Serial\Unit]

Object Type: [CDM v20\Modbus v31\Serial\Unit]

Object Type: [OSM v20\Modbus v30\Serial\Unit]

Object Type: [CDM v20\Modbus v30\Serial\Unit]

The Serial Unit object is used to optionally configure the address of a Modbus serial device on the network.

Description	Reference	Type
<b>Label</b> Label displayed when scanning the Modbus system	L	Obj\Text: 20 chars; Adjustable
<b>Serial Address</b> Modbus over serial device address	A	Obj\Num: 1...247; Adjustable
<b>Device Type</b> Leave blank to view default Modbus objects for a device. Refer to <i>Application</i> <i>Note</i> for other types available	DT	Obj\Text: 20 chars; Adjustable

# Modbus Serial Server Setup

Object Type: *[OSM v20\Modbus v31\Serial\Server]*

Object Type: *[CDM v20\Modbus v31\Serial\Server]*

Object Type: *[OSM v20\Modbus v30\Serial\Server]*

Object Type: *[CDM v20\Modbus v30\Serial\Server]*

The Modbus Serial Server Setup object is used to configure the address of the North device on the Modbus serial network.

Description	Reference	Type
<b>Address on Modbus network</b> Set to a unique address on the RS485 network	ADDR	Obj\Num; 1...247; Adjustable
<b>Security</b> Configure privilege levels to control read and adjust access to Essential Data and Extra Data from a Modbus serial client	S	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\Privs]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\Privs]</i>
<b>Register List</b> List of registers made available from Essential Data to a Modbus serial client. Useful for documentation.	RL	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\RegList]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\RegList]</i>

# Security

Object Type: [OSM v20\Modbus v31\Privs]

Object Type: [CDM v20\Modbus v31\Privs]

Object Type: [OSM v20\Modbus v30\Privs]

Object Type: [CDM v20\Modbus v30\Privs]

## Security Areas and Levels

Within the North security model, there are eight security areas. Security areas could be actual areas in a building, but are normally functional areas – for example, ‘environmental control’ and ‘North engineering’ areas would allow a user to have different privileges in controlling set points and engineering Commanders.

Typically, a user is assigned a privilege level in each of the eight areas. The level is in the range zero to seven, seven being the most powerful. When a user wishes to pass a door, his/her privilege level in the door’s area is checked against the minimum required for that area – and then either allowed to pass, or rejected.

The engineer must decide the use of the eight areas. The engineer must also decide the power of the privilege levels. Most systems use only a few levels per area: 0=None, 1=Guest, 2=User, 7=Administrator.

As an example, imagine a page of values in Essential Data. The page needs a user to have a minimum privilege level of 2 in area 1 before it can be viewed. The page is available in a Web browser that checks users with a security database. User A has privilege level 7 in area 1 – she can view the page. User B has privilege level 5 in area 1 – he can also view the page. User C has privilege level 1 in area 1 – she cannot view the page.

The example continues: within this page of values in Essential Data is a temperature set point object. Users need a minimum privilege level of 6 in area 1 to adjust it – therefore User A can adjust the set point, but User B cannot.

## Specifying Access Security

Essential Data and Extra Data have Access Security objects to control who can view a page, and who can adjust an adjustable object.

Each Access Security object has a two-digit value. Each controls the access to a particular feature - such as viewing the page or adjusting the value. The two-digit value is made up of the area digit (1-8), followed by the minimum privilege level (1-7) – for example, if the minimum privilege level is 6 in area 2, then the two-digit value is 26. If the value is 00, then no security checks are made.

## Modbus Driver

The Security object contains a privilege level for each of the eight security areas, representing a virtual user. The Modbus driver uses these to control access to Essential Data and Extra Data when reading or adjusting a value.

Description	Reference	Type
<b>Privilege Level in Area x</b> The area, x, can be in the range 1...8	Px	Obj\Num; Adjustable; Range: 0...7

## Register List

Object Type: *[CDM v20\Modbus v31\RegList]*

Object Type: *[OSM v20\Modbus v31\RegList]*

The Register List object contains the list of available Modbus registers presented from the North device's Essential Data. This list is provided for documentation and fault-finding purposes.

Export the Register List to a CSV file using engineering software. Use this CSV to provide a list of configured registers for third-party integration.

Description	Reference	Type
<b>Registers Available</b> Count of maximum objects available from Essential Data and Extra Data	EDC	Obj\Num
<b>Register x</b> The register address, x, can be in the range 0...639 on Commander, and 0...1279 on ObSys. If Extra Data is used, the register address range is extended to 1663 on Commander, and 2303 on ObSys.	REGx	Fixed Container: On the Commander platform this will be <i>[CDM v20\Modbus v31\RegList\Addr]</i> On the ObSys platform this will be <i>[OSM v20\Modbus v31\RegList\Addr]</i>

## Register

Object Type: *[CDM v20\Modbus v31\RegList\Addr]*

Object Type: *[OSM v20\Modbus v31\RegList\Addr]*

A Register contains a single register address, documenting how the value from Essential Data is presented as a Modbus register.

Description	Reference	Type
<b>Label</b> Label from Essential Data	L	Obj\Text
<b>Format</b> Format of the register value	F	Obj\Text Values: Signed, Unsigned, n/a
<b>Adjustable</b> Indicates if the register can be written to using Modbus function codes 05 or 06	A	Obj\NoYes
<b>Register Value</b> Value converted to a 16-bit Modbus value	V	Obj\Num: 0...65535
<b>Note</b> Additional information about the register value. This may contain the units, a list of Enum values, or a decoded Float value	N	Obj\Text



# Advanced Settings

Object Type: [OSM v20\Modbus v31\Advanced]

Object Type: [CDM v20\Modbus v31\Advanced]

Object Type: [OSM v20\Modbus v30\Advanced]

Object Type: [CDM v20\Modbus v30\Advanced]

The Advanced Settings object contains the following advanced configuration objects:

Description	Reference	Type
<b>Maximum Requests</b> Maximum number of simultaneous requests from the interface to connected Modbus client devices	MR	Obj\Num: 1...3; Adjustable Default: 3
<b>Reply Timeout (ms)</b> Maximum time to wait from sending a request to receiving a reply from a Modbus device	TO	Obj\Num: 250...2000; Adjustable Default: 2000ms
<b>Inter-Frame Delay (ms)</b> On Modbus serial connections, time to wait between message frames. As a minimum, this value must be the time taken to send 3.5 characters.	T3	Obj\Num: 5...1000; Adjustable Default: 50ms
<b>TCP Client Keep-Alive (s)</b> On TCP Client connections, the maximum time between requests to keep the connection established	KA	Obj\Num: 10...1800; Adjustable Default: 15s
<b>Cache Life (s)</b> When decoding a single register of a multi-register value (decode 'R'), the driver caches the multi-register value for Cache Life (in seconds)	CL	Obj\Num: 1...900; Adjustable Default: 25s
<b>Register Byte Order</b> The Modbus protocol uses a big-endian byte order. For non-standard client devices use this object to reverse the byte order. Refer to <i>Application Note</i> or contact support for help. Server operation is always big-endian	BO	Obj\Enum; Adjustable Values: 0=Big-endian, 1=Little-endian Default: big-endian
<b>Formula y</b> User defined mathematical formula, used in decoding values from device. The formula number, y, is in the range 1...20	Fy	Fixed Container: <a href="#">[Standard\AMFormula]</a>
<b>Debug Enable</b> This will store additional debug information in the record file. Use this option only when instructed by North Support	DE	Obj\NoYes; Adjustable

# Formula Setup

Object Type: *[Standard\AMFormula]*

A standard formula setup is a maths module that allows values to be converted into engineering units.

This module allows a simple formula to be applied to a Modbus register value so that the resulting object value contains a meaningful value.

The module can convert the number into an object value using the formula:

$$\text{real-value} = (\text{M} \times \text{raw-value}) + \text{A}$$

The formula values M and A are engineer-defined. When writing, the module applies the formula below:

$$\text{raw-value} = (\text{real-value} - \text{A}) / \text{M}$$

## Example

A register value contains a temperature value, where value 0 = -50°C and value 65535 = +50°C.

If A is set to -50 and M=0.0015259, then the formula would be:

$$\text{temperature} = (0.0015259 \times \text{register-value}) + -50$$

So, the following temperatures can be calculated from the register values:

Register value	Temperature
0	-50°C
32767	0°C
49150	25°C
65535	50°C

The module contains the following objects:

Description	Reference	Type
Addition value	A	Obj\Float; Adjustable
Multiplication value	M	Obj\Float; Adjustable

# Modbus System

Object Type: *[Modbus]*

The Modbus system contains objects to access the Modbus client devices available.

Description	Reference	Type
<b>Unit <i>x</i></b> The unit address, <i>x</i> , can be in the range 1...30.	U <i>x</i>	Fixed container, one of the following: Default Modbus Device <a href="#">[Modbus\Default]</a> If <i>Device Type</i> is configured then the container will be of the type [Modbus\Device Type]. See <a href="#">Modbus TCP Client Setup</a>
<b>Serial Address <i>y</i></b> The serial address, <i>y</i> , can be in the range 1...247.	A <i>y</i>	Fixed container, one of the following: Default Modbus Device <a href="#">[Modbus\Default]</a> If <i>Device Type</i> is configured then the container will be of the type [Modbus\Device Type]. See <a href="#">Modbus Serial Client Setup</a>

# Default Modbus Device

Object Type: [Modbus\Default]

A default Modbus device contains a generic list of objects that enable you to access the values in a device. Use this with the device manufacturer's register list. For a full description of Modbus values and how to decode them, refer to *Operation as a Modbus Client* earlier in this document.

## Frequently Used Objects

Here we list a summary of the most frequently used decode objects. Refer to the *Notes* section below for additional information, and *Full Object List* below for the complete list of objects available.

Description	Reference	Type
<b>Coil <i>r</i> – State</b> The digital output, <i>r</i> , is in the range 0...65535 (see note 1)	Cr.A	Obj\OffOn; Adjustable
<b>Discrete Input <i>r</i> – State</b> The digital input, <i>r</i> , is in the range 0...65535 (see note 1)	Ar.A	Obj\OffOn
<b>Holding Register <i>r</i> – Unsigned 16-bit Integer</b> The register, <i>r</i> , is in the range 0...65535 (see note 1).	Dr.B	Obj\Num; Range: 0...65535; Adjustable
<b>Holding Register <i>r</i> – Unsigned 16-bit (x10)</b>	Dr.B26	Obj\Float; Range: 0...6553.5; Adjustable
<b>Holding Register <i>r</i> – Unsigned 16-bit (x100)</b>	Dr.B27	Obj\Float; Range: 0...655.35; Adjustable
<b>Holding Register <i>r</i> – Unsigned 16-bit (x1000)</b>	Dr.B28	Obj\Float; Range: 0...65.535; Adjustable
<b>Holding Register <i>r</i> – Signed 16-bit Integer</b>	Dr.C	Obj\Num; Range: -32768...32767; Adjustable
<b>Holding Register <i>r</i> – Signed 16-bit (x10)</b>	Dr.C26	Obj\Float; Range: -3276.8...3276.7; Adjustable
<b>Holding Register <i>r</i> – Signed 16-bit (x100)</b>	Dr.C27	Obj\Float; Range: -327.68...327.67; Adjustable
<b>Holding Register <i>r</i> – Signed 16-bit (x1000)</b>	Dr.C28	Obj\Float; Range: -32.768...32.767; Adjustable
<b>Holding Register <i>r</i> – IEEE Float</b> Single precision, stored in two registers (MSW, LSW)	Fr.J	Obj\Float; Adjustable
<b>Holding Register <i>r</i> – Unsigned 32-bit Integer</b> Stored in two registers (MSW,LSW) (see note 2)	Fr.B	Obj\Num; Range: 0...4294967295; Adjustable
<b>Holding Register <i>r</i> – Signed 32-bit Integer</b> Stored in two registers (MSW,LSW) (see note 2)	Fr.C	Obj\Num; Range: -2147483648...2147483647; Adjustable
<b>Holding Register <i>r</i> – Unsigned 64-bit Integer</b> Stored in four registers (MSW...LSW) (see note 2)	Hr.B	Obj\Text; Range: 0...18446744073709551615; Adjustable
<b>Holding Register <i>r</i> – Signed 64-bit Integer</b> Stored in four registers (see note 2)	Hr.C	Obj\Text; Range: -923372036854775808...923372036854775807; Adjustable
<b>Holding Register <i>r</i> – IEEE Double Float</b> Double precision, stored in four registers (MSW... LSW) (see note 2)	Hr.J	Obj\Text; Adjustable

Description	Reference	Type
<b>Holding Register <math>r</math> – Bit <math>b</math></b> The bit number, $b$ , can be in the range 0...15. Refer to <i>Single Bit of Register</i> section earlier in this manual	Dr.Kb	Obj\OffOn; Adjustable
<b>Input Register <math>r</math> – Unsigned 16-bit Integer</b> The register, $r$ , is in the range 0...65535 (see note 1)	Br.B	Obj\Num; Range: 0...65535
<b>Input Register <math>r</math> – Unsigned 16-bit (x10)</b>	Br.B26	Obj\Float; Range: 0...6553.5
<b>Input Register <math>r</math> – Unsigned 16-bit (x100)</b>	Br.B27	Obj\Float; Range: 0...655.35
<b>Input Register <math>r</math> – Unsigned 16-bit (x1000)</b>	Br.B28	Obj\Float; Range: 0...65.535
<b>Input Register <math>r</math> – Signed 16-bit Integer</b>	Br.C	Obj\Num; Range: -32768...32767
<b>Input Register <math>r</math> – Signed 16-bit (x10)</b>	Br.C26	Obj\Float; Range: -3276.8...3276.7
<b>Input Register <math>r</math> – Signed 16-bit (x100)</b>	Br.C27	Obj\Float; Range: -327.68...327.67
<b>Input Register <math>r</math> – Signed 16-bit (x1000)</b>	Br.C28	Obj\Num; Range: -32.768...32.767
<b>Input Register <math>r</math> – IEEE Float</b> Single precision, stored in two registers (MSW,LSW)	Nr.J	Obj\Float
<b>Input Register <math>r</math> – Unsigned 32-bit Integer</b> Stored in two registers (MSW,LSW) (see note 2)	Nr.B	Obj\Num; Range: 0...4294967295
<b>Input Register <math>r</math> – Signed 32-bit Integer</b> Stored in two registers (MSW,LSW) (see note 2)	Nr.C	Obj\Num; Range: -2147483648...2147483647
<b>Input Register <math>r</math> – Unsigned 64-bit Integer</b> Stored in four registers (MSW...LSW) (see note 2)	Pr.B	Obj\Text; Range: 0...18446744073709551615
<b>Input Register <math>r</math> – Signed 64-bit Integer</b> Stored in four registers (MSW...LSW) (see note 2)	Pr.C	Obj\Text; Range: -923372036854775808...923372036854775807
<b>Input Register <math>r</math> – IEEE Double Float</b> Double precision, stored in four registers (MSW...LSW) (see note 2)	Pr.J	Obj\Text
<b>Input Register <math>r</math> – Bit <math>b</math></b> The bit number, $b$ , can be in the range 0...15. Refer to <i>Single Bit of Register</i> section earlier in this manual	Br.Kb	Obj\OffOn

## Notes

1. The discrete input, coil, or register number,  $r$ , is in the range 0...65535. Manufacturers sometimes document the register number in the range 1...65536. To rescale these in the 0 to 65535 range from the Modbus protocol, you will need to subtract 1. For example, register 100 becomes 99.
2. Large numbers: 64-bit and 32-bit values can contain up to 20 significant figures. Numbers this size are ok for displaying to a user, but may be too large to perform accurate maths functions. These values can be read in blocks of six significant figures by appending the object reference with a block number. Block 1 reads the six least significant figures, block 2 the next six significant figures, etc.  
For example, if object 'H1.B' reads the 64-bit value '6744073709551615', then object 'H1.B.1' will read the least six significant figures '551615', object 'H1.B.2' the value '073709', and object 'H1.B.3' the value '6744'.

You can also use a formula with a block number. The object has the format 'H1.B28.1'. This will apply the formula first then access the requested block of six significant figures. If the formula uses a divisor, then the value will be formatted to three decimal places. For example, object 'H1.B28.1' will read the value '551.615'.

## Full Object List

The Modbus driver contains the following objects:

Description	Reference	Type
<b>Function <i>f</i>, Entry <i>r</i> – Decode <i>d</i></b> The function, <i>f</i> , is in the range A...U (see note 3) The entry, <i>r</i> , is in the range 0...65535 (see note 1) The decode, <i>d</i> , is in the range A...J, L, or M, or O...P (see also note 4). Refer to <a href="#">Value Decoding</a> earlier in this manual. For 2 and 4 register values, see note 5.	<i>Fr.d</i>	The value type is dependent on the function, <i>f</i> , and decode, <i>d</i> . Adjustable for Coils and Holding Registers
<b>Function <i>f</i>, Entry <i>r</i> – Decode <i>d</i>, Formula <i>z</i></b> As above, but with formula, <i>z</i> , applied (see note 2)	<i>fr.dz</i>	The value type is dependent on the function, <i>f</i> , decode, <i>d</i> , and formula, <i>z</i> . Adjustable for Coils and Holding Registers
<b>Function <i>f</i>, Entry <i>r</i> – Bit <i>b</i></b> Refer to <a href="#">Value Decoding: Single Bit of Register</a> earlier in this manual. The function, <i>f</i> , is in the range D...L, B, N...T (see note 3) The entry, <i>r</i> , is in the range 0...65535 (see note 1) The bit number, <i>b</i> , can be in the range 0...15.	<i>Fr.Kb</i>	Obj\OffOn; Adjustable for Holding Registers
<b>Function <i>f</i>, Entry <i>r</i> – Bit Mask <i>m</i></b> Refer to <a href="#">Value Decoding: Bit Mask</a> earlier in this manual. The function, <i>f</i> , is in the range D...F, B, N (see note 3) The entry, <i>r</i> , is in the range 0...65535 (see note 1) The bit mask, <i>m</i> , is a number in the range 0...65535.	<i>Fr.Lm</i>	Obj\Num
<b>Function <i>f</i>, Entry <i>r</i> – Bit Mask <i>m</i>, Formula <i>z</i></b> As above, but with formula, <i>z</i> , applied (see note 2)	<i>fr.Lm.z</i>	Obj\Float
<b>Function <i>f</i>, Entry <i>r</i> – Bit Mask <i>m</i>, Shift <i>s</i></b> Refer to <a href="#">Value Decoding: Bit Mask with Bit Shift</a> earlier in this manual. The function, <i>f</i> , is in the range D...F, B, N (see note 3) The entry, <i>r</i> , is in the range 0...65535 (see note 1) The bit mask, <i>m</i> , is a number in the range 0...65535. The bit shift, <i>s</i> , is a number in the range 1...32.	<i>Fr.Nm.s</i>	Obj\Num
<b>Function <i>f</i>, Register <i>r</i></b> Read a multi-register value, and decode a single register as an unsigned integer. Refer to <a href="#">Value Decoding: Single Register of Multi-Register</a> earlier in this manual. The function, <i>f</i> , is in the range E...L, N...T (see note 3) The register index, <i>r</i> , is in the range 1...16. Available in driver version 3.1 onwards.	<i>Fr.R</i>	Obj\Num: 0...65535

## Notes

1. The discrete input, coil, or register number,  $r$ , is in the range 0...65535. Manufacturers sometimes document the register number in the range 1...65536. To rescale these in the 0 to 65535 range from the Modbus protocol, you will need to subtract 1. E.g. Register 100 becomes 99.
2. An optional formula number,  $z$ , may be applied where indicated above. The formula number is in the range 1...40, where 1...20 refer to user-defined formula, and 21...40 are fixed as follows:

Formula	Multiply	Add	Formula	Multiply	Add
21	10	0	31	2	0
22	100	0	32	5	0
23	1000	0	33	0.2	0
24	10000	0	34	0.5	0
25	100000	0	35	0.05	0
26	0.1	0	36	0.005	0
27	0.01	0	37	0.000001	0
28	0.001	0	38	1	0
29	0.0001	0	39	1	0
30	0.00001	0	40	Four quadrant power factor (Float decode only)	

3. The function,  $f$ , is the Modbus function or command and can be in the range A...U. Refer to the [Function Codes](#) section earlier in this document.

Driver Function	Table	Action	Function Code (read)	Function Code (write)
C	Coils	Read/Write digital output	01	05
A	Discrete Inputs	Read digital input	02	
D	Holding Registers	Read/Write 1 output register	03	06
E	Holding Registers	Read/Write 1 output multi-registers	03	16
F	Holding Registers	Read/Write 2 output multi-registers	03	16
G	Holding Registers	Read/Write 3 output multi-registers	03	16
H	Holding Registers	Read/Write 4 output multi-registers	03	16
I	Holding Registers	Read/Write 6 output multi-registers	03	16
J	Holding Registers	Read/Write 8 output multi-registers	03	16
K	Holding Registers	Read/Write 10 output multi-registers	03	16
L	Holding Registers	Read/Write 16 output multi-registers	03	16
B	Input Registers	Read 1 input register	04	
N	Input Registers	Read 2 input multi-registers	04	
O	Input Registers	Read 3 input multi-registers	04	
P	Input Registers	Read 4 input multi-registers	04	
Q	Input Registers	Read 6 input multi-registers	04	
R	Input Registers	Read 8 input multi-registers	04	
S	Input Registers	Read 10 input multi-registers	04	
T	Input Registers	Read 16 input multi-registers	04	
U	Coils	Read/Write 1 digital output	01	15



4. The decode, *d*, is in the range A...Q. Refer to the [Value Decoding](#) section earlier in this document.

Decode	Use	Object Type
A	Digital State	Obj\OffOn
B	Unsigned Integer	Obj\Num
C	Signed Integer	Obj\Num
D	BCD in lower nibbles only	Obj\Num
E	BCD in register	Obj\Num
F	ASCII in LSB	Obj\Text
G	ASCII string	Obj\Text
H	Unsigned Integer in LSB	Obj\Num
I	Unsigned Integer in MSB	Obj\Num
J	IEEE Float (MSW, LSW order)	Obj\Float
M	IEEE Float (LSW, MSW order)	Obj\Float
K	Single bit of register	Obj\OffOn
L	Bit Mask	Obj\Num
N	Bit Mask with Bit Shift	Obj\Num
O	Unsigned Integer (LSW, MSW order)	Obj\Num
P	Signed Integer (LSW, MSW order)	Obj\Num
Q	Special Decode – Ask North	<various>
R	Unsigned Integer in Single Register	Obj\Num

5. Large numbers: 64-bit and 32-bit values can contain up to 20 significant figures. Numbers this size are ok for displaying to a user but may be too large to perform accurate maths functions. These values can be read in blocks of six significant figures by appending the object reference with a block number. Block 1 reads the six least significant figures, block 2 the next six significant figures, etc.

For example, if object 'H1.B' reads the 64-bit value '6744073709551615', then object 'H1.B.1' will read the least six significant figures '551615', object 'H1.B.2' the value '073709', and object 'H1.B.3' the value '6744'.

You can also use a formula with a block number. The object has the format 'H1.B28.1'. This will apply the formula first then access the requested block of six significant figures.

If the formula uses a divisor, then the value will be formatted to three decimal places. For example, object 'H1.B28.1' will read the value '551.615'.

# Appendix A: Modbus Integration Summary

When the Modbus TCP Server is enabled, or Modbus Serial Mode is set to 'Server', the North device presents values from its database, Essential Data, as Modbus values.

## Modbus over TCP/IP

The default TCP port is 502. The North device will respond to requests with a Modbus unit identifier of 255, 0, or 1.

## Modbus over Serial-line

The engineer configuring the North device can provide the Modbus address along with the baud rate, data parity, and stop bits (default 9600, even parity, 1 stop bit).

Modbus RTU operating mode is supported.

## Function Codes

The North device supports the following Modbus functions codes. Some Modbus messages can contain a quantity of items, and the table shows the range of quantities supported for each function:

Function Code	Action	Quantity
01	Read Coils	1 to 255
02	Read Discrete Inputs	1 to 255
03	Read Holding Registers	1 to 125
04	Read Input Registers	1 to 125
05	Write Single Coil	n/a
06	Write Single Holding Register	n/a
15	Write Multiple Coils	1 to 255
16	Write Multiple Holding Registers	1 to 123

## Register Address

The North device has only one data table – Essential Data – and so maps all Modbus requests to a single register, input, or coil address range for all functions – 0...639 on Commander, and 0...1279 on ObSys.

This Modbus register address maps to an Essential Data page/object reference. For example, register 0 maps to the first object 'P1.O1', register 1 to 'P1.O2', etc. However, an easier way to see how register addresses are used is to look at the exported Register List.

## Register List

The engineer configuring the North device can export a Register List. This CSV file contains columns with the following:

- Register address – 'REG' followed by decimal register address, starting at 0
- Label
- Format – format of value: Unsigned or Signed; N/A: not available via Modbus
- Adjustable – indicates if the register value can be written (e.g. holding register)
- Register Value – current value converted to a 16-bit Modbus register value
- Note – additional information: units, available values (enum), etc.

Example exported register list:

Index	Object	Label	Format	Adjustable	Register Value	Note
1	REG0	Example - Counter	Unsigned	No	2025	
2	REG1	Example - State	Unsigned (enum)	No	1	0=Off,1=On
3	REG2	Example - Temperature	Signed x10	No	185	Value: 18.5 °C
4	REG3	Example - Humidity	Unsigned	No	58	%rh
5	REG4	Example - Setpoint	Signed	Yes	19	°C
6	REG5	Example - Fan	Unsigned (enum)	Yes	3	0=Off,1=Low,3=Auto
7	REG6	Example - not available	N/A	No	0	

## Register Value

A register value may be accessed using any of the supported Modbus function codes.

The Essential Data value is translated to a Modbus register or state depending which Modbus Table is specified and the value format configured:

Modbus Table	Value Format	
	Unsigned	Signed
<b>Input Register, Holding Register</b>	16-bit unsigned register value in the range 0...65535.	16-bit signed value in the range -32768...32767
		The format field may indicate the value is scaled, e.g. 'Signed x 10'. In this case, the register value should be converted to a floating point number (by dividing by 10). The note field includes a converted value.
<b>Discrete Input, Coil</b>	Binary on/off state. If the value is zero (0) then an 'off' state is returned, any other value returns an 'on' state	

## Checking Values

The North device has a local web-server, browse to the device's IP address to view values.

# Driver Versions

Version	Build Date	Details
1.0	10/9/2012	Driver released (renamed from JBus).
1.0	29/8/2013	Change Modbus system scan to detect online devices. Default baud rate to 9600 on initialisation Address Start and Count now initialised
1.0	6/9/2013	Fixed problem with bit write
1.1	7/2/2014	Moved objects AC, AS, TXB and BO to advanced setup object (A) Add Intelligent scan object to disable new scan method (A.IS) Add exception device typelist (A.Ux.A, A.Ux.DT) Default TXB to 10ms on initialisation
1.1	19/5/2014	On initialisation, default baud rate to 19200, and E81 byte format
2.0	8/2/2016	Added Reply Timeout object (TO) to advanced setup. 64-bit value support added Decode types H & I, now read before writing value. Added formulas 37 and 40 Added ability to read in blocks of 6 sig figs, for 32-bit and 64-bit values
2.0	18/6/2018	Fix reading 64-bit float values when 0 (previously returned blank)
2.0	17/8/2018	Added function code 'U' to support Modbus function code 15
3.0	2/12/2019	Combined existing Modbus, ModbusTCP, and ModbusSlave drivers in this new version. Added decode 'N' for bit mask with bit shift.
3.0	03/02/2020	Fix documentation of input/holding multi-register counts (5,6,7,8 should be 6,8,12,16)
3.0	01/06/2020	Added decode 'O' and 'P' for LSW,MSW order. Added undocumented special decode mechanism 'Q' Changed multi-register count from 12 to 10 registers. Resolved issue on rounding floating-point numbers when writing
3.0	16/11/2022	TCP server is no longer enabled by default
3.0	01/03/2023	Fix issue when decoding 4-register values (function 'H')
3.1	21/01/2025	Add decode 'R', decoding single register of a multi-register value. Add Cache Life driver object. Used to cache a multi-register value with decode R. Add TCP Client Keep-Alive driver object, previously fixed to 15s. On Commander platform, add support for requests to client at loopback address (127.0.0.1) Modbus Server register value format changed for float (signed) and num (unsigned). Previously the format varied between unsigned/signed based on a -ve value. Update Register List to improve CSV export and update Appendix A. Fix issue with serial request/response mismatch on failed responses.

## Next Steps...

If you require help, contact support on 01273 694422 or visit [www.northbt.com/support](http://www.northbt.com/support)



North Building Technologies Ltd  
+44 (0) 1273 694422  
support@northbt.com  
www.northbt.com

This document is subject to change without notice and does not represent any commitment by North Building Technologies Ltd.

ObSys and Commander are trademarks of North Building Technologies Ltd. All other trademarks are property of their respective owners.

© Copyright 2025 North Building Technologies Limited.

Author: JF  
Checked by: TM

Document issued 22/01/2025.