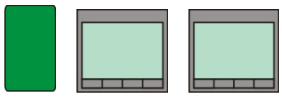
The Modbus Driver



The Modbus driver connects to a wide range of equipment supporting the Modbus over serial line and JBus protocols. Compatible slave devices includes power meters, generators, and PLCs. Available for Commander and ObSys.

This document relates to Modbus driver version 1.0, 1.1 and 2.0

Please read the *Commander Manual* or *ObSys Manual* alongside this document, available from *www.northbt.com*

Contents

Compatibility with the Modbus System

The Modbus driver allows North to interface with a wide range of equipment supporting the Modbus over serial line and JBus protocols. Compatible equipment includes power meters, generators, and PLCs.

Modbus over serial line uses a master-slave model. The Modbus driver must be the only master device on the network, capable of requesting values from a slave device. All other connected Modbus equipment must be slave devices, capable of responding to the master device.

The driver supports devices with an RS485 or RS232 connection. Using RS485, a network of Modbus devices can be connected (Fig. 1). Using RS232, only a single Modbus device can be connected (Fig. 2).

At least 32 devices are supported on an RS485 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 supports up to 247 devices.

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

Two alternative Modbus drivers are also available. The ModbusTCP driver provides support for Modbus over TCP/IP networks with client and server interfaces, and the ModbusDev driver provides a Modbus over serial line slave device interface.

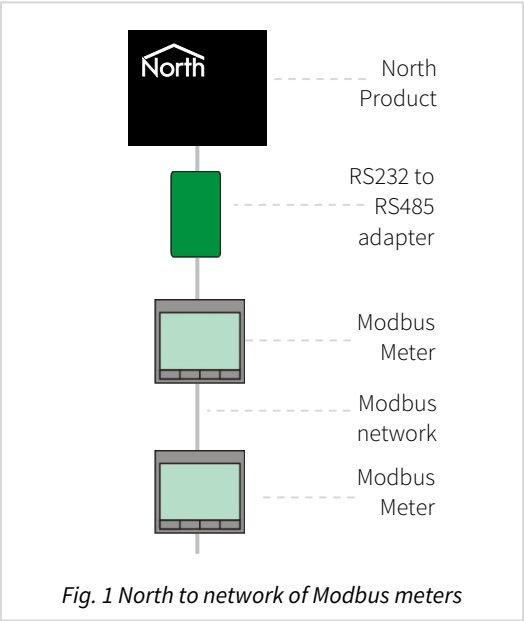


Fig. 1 North to network of Modbus meters

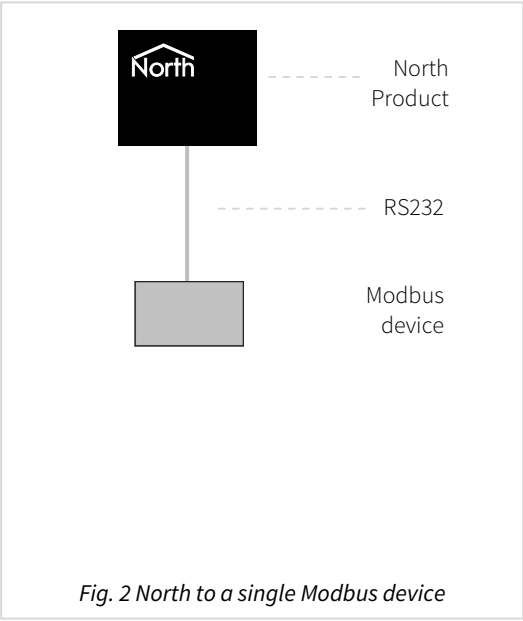


Fig. 2 North to a single Modbus device

Equipment

Many different types of Modbus and JBus 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, General Electric, Merlin Gerin, Mitsubishi, Schneider, Socomec, Stulz, Swegon, Tyco, plus many more.

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

Values

Depending on the type of Modbus equipment, each device can typically have the following 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.

Prerequisites

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, Modbus operating mode, TX delay, and RS485 isolation.

If an *Application Note* is not available for your device, 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.)

All Modbus devices must be configured with a unique address on the network, and the following common parameters: Modbus operating mode (RTU or ASCII), baud rate, byte format.

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

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.

Making the Cable

RS485 Devices

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

Using the RS485 cable specification (Fig. 3), connect the RS485 adapter to the Modbus device network.

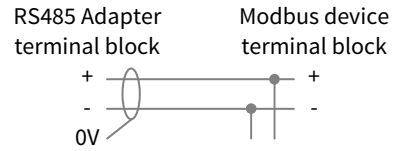


Fig. 3 RS485 adapter to Modbus cable

RS485 adapters are available from North, order code MISC/RS232/485.

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.

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 that 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.

Setting up the Driver

To set up the driver, follow these steps:

- → Navigate to the **Modbus Setup** object. For example, if you started interface 1 with the driver earlier, then the object reference will be 'M1'
- → Set the **RS232 Com Port** to the port number of the North device you are connecting to Modbus
- → Set **Baud Rate** to match the Modbus devices, typically 9600, 19200 or 38400 baud
- → Set the **Byte Format** to the parity, data bits and stop bits configured in the Modbus devices
- → Set the **Operating Mode** to match the Modbus devices, RTU or ASCII
- → Set **Address Count** to the largest address on the Modbus network

→ Follow additional steps on the *Application Note*, if available

Checking Communications

You can check the interface is communicating by scanning the **Modbus System** and viewing values within an address.

Understanding Modbus

Data Model

A Modbus device stores data using four primary tables.

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, internally calculates 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. 4).

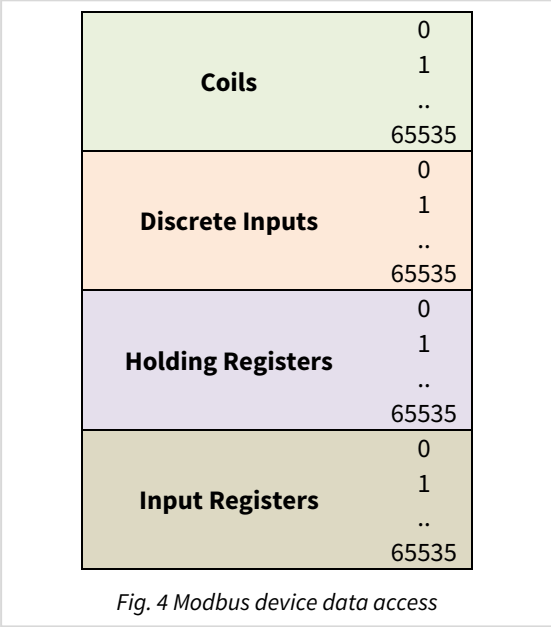


Fig. 4 Modbus device data access

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 data values
- Register addresses are often documented in the range 1 to 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.

Function Codes

Values from a particular table are read and adjusted using different Modbus function codes. A device may only support some of these codes. The driver function is used as part of the object reference described later.

Table	Action	Function Code (read)	Function Code (write)	Driver Function
Coils	Read/Write digital output	01	05	C
Discrete Inputs	Read digital input	02		A
Holding Registers	Read/Write single output register	03	06	D
Holding Registers	Read/Write multiple output registers	03	16	E to L
Input Registers	Read single input register	04		B
Input Registers	Read multiple input registers	04		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
- IEEE floating point values
- 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 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)

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.

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)

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.

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 Sum(V): 80+8+1 = 89

	Unused nibble				Lower nibble				Unused nibble				Lower nibble			
Bit value (V)					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	1

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 number from a register value. Bits are numbered starting with the most significant byte of the first register, as shown in the example. 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). Bits 8, 11, 14, 3, and 6 have the value 1. All other bits have the value 0.

Bit number	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: Sum(V) = 64+32+16+8 = 120. So, Register AND Mask = 18505 AND 120 = 72.

We could additionally use a formula to rescale the value, i.e. 72 x 0.125 = 9

Bit value (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

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 an Address (A1), which contains an Input register (B2.B). Therefore, the object reference will be 'S1.A1.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.A1.B2.B) – therefore the complete object reference is 'IP.CDIP.S1.A1.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
Modbus Module Set up the Modbus driver, started on interface *c* (*c* is the interface number)	M*c*	Fixed Container: On the Commander platform this will be *[CDM v20\Modbus v20]* On the ObSys platform this will be *[OSM v20\Modbus v20]*
Modbus System Access Modbus system connected to interface *c* (*c* is the interface number)	S*c*	Variable Container: *[Modbus]*

Modbus Driver Setup

Object Type: *[OSM v20\Modbus v20]*
Object Type: *[CDM v20\Modbus v20]*
Object Type: *[OSM v20\Modbus v11]*
Object Type: *[CDM v20\Modbus v11]*
Object Type: *[OSM v20\Modbus v10]*
Object Type: *[CDM v20\Modbus v10]*

The Modbus driver contains the following objects:

Description	Reference	Type
RS232 COM Port	RS.COM	Obj\Num: 1…8; Adjustable
Baud Rate	RS.BR	Obj\Num; Adjustable; Default: 19200 Range: 1200, 2400, 4800, 9600, 19200 or 38400
Byte Format Sets the parity, data bits, and stop bits	RS.BF	Obj\ENum: 0…11; Adjustable; Default: E81 See note 1
Operating Mode Modbus framing	OM	Obj\Enum: 0…1; Adjustable; Default: RTU Values: 0=RTU Framing, 1=ASCII Framing
System Label Label displayed when scanning the system	DL	Obj\Text: 20 chars; Adjustable
Device Type Leave blank to view default Modbus objects for a device. Refer to *Application Note* for other device types available. To specify a different device type for a particular address use the Unit Setup object (A.U*x*)	DT	Obj\Text: 20 chars; Adjustable
Advanced Setup Available in driver version 1.1 and later	A	Fixed Container: On the Commander platform this will be *[CDM v20\Modbus v20\Advanced]* On the ObSys platform this will be *[OSM v20\Modbus v20\Advanced]*
Formula *x* User defined mathematical formula, used in decoding values from device. The formula number, *x*, is in the range 1…20	F*x*	Fixed Container: *[Standard\AMFormula]*

Notes

1 Byte format can have the following values:

Value	Parity	Data bits	Stop bits	Notes
0	None	8	1	In RTU Framing mode, use N82 rather than N81
1	None	8	2	
2	None	7	1	Only use in ASCII Framing mode
3	None	7	2	Only use in ASCII Framing mode
4	Odd	8	1	
5	Odd	8	2	
6	Odd	7	1	Only use in ASCII Framing mode
7	Odd	7	2	Only use in ASCII Framing mode
8	Even	8	1	
9	Even	8	2	
10	Even	7	1	Only use in ASCII Framing mode
11	Even	7	2	Only use in ASCII Framing mode

Modbus Advanced Setup

Object Type: *[OSM v20\Modbus v20\Advanced]*
Object Type: *[CDM v20\Modbus v20\Advanced]*
Object Type: *[OSM v20\Modbus v11\Advanced]*
Object Type: *[CDM v20\Modbus v11\Advanced]*

The Modbus driver contains the following advanced configuration objects:

Description	Reference	Type
Address Start Address of lowest numbered device. Used when scanning the Modbus system	AS	Obj\Num: 1…247; Adjustable
Address Count Use with Address Start to provide the last address to scan from the Modbus System	AC	Obj\Num: 0…247; Adjustable
TX Delay (ms) Provides a delay between response and next request. Not usually required. Refer to *Application Note* or contact support for help	TXB	Obj\Num: 0…1000; Adjustable
Reply Timeout (ms) Maximum time to wait from sending a request to receiving a reply from the Modbus device. Typically this should be left at the default value of 2000ms.	TO	Obj\Num: 250…2000; Adjustable
Byte Order The Modbus protocol uses a big-endian byte order. For non-standard implementations use this object to reverse the byte order. Refer to *Application Note* or contact support for help	BO	Obj\ENum; Adjustable Values: 0=Standard, 1=Reverse TX only, 2=Reverse RX only, 3=Reverse TX/RX
Intelligent Scan With this option enabled, the driver attempts to detect if a device is present when scanning the Modbus object	IS	Obj\OffOn; Adjustable
Half-Float Format Sets conversion for 16-bit half-float values	HFF	Obj\ENum; Adjustable Values: 0=IEEE, 1=Toshiba
Unit *x* Setup Specify addresses with a different device type then specified in object DT. The unit number, *x*, is in the range 1…10	U*x*	Fixed Container: On the Commander platform this will be *[CDM v20\Modbus v11\Unit]* On the ObSys platform this will be *[OSM v20\Modbus v11\Unit]*

Modbus Unit Setup

Object Type: *[OSM v20\Modbus v20\Unit]*
Object Type: *[CDM v20\Modbus v20\Unit]*
Object Type: *[OSM v20\Modbus v11\Unit]*
Object Type: *[CDM v20\Modbus v11\Unit]*

The Device Type object (DT) sets a default device type for all connected Modbus devices. Use the Modbus Unit Setup object to specify any devices with a different device type.

Specify the Modbus address and its device type using the objects here.

Description	Reference	Type
Address Address of Modbus device	A	Obj\Num: 0…247; Adjustable
Device Type Leave blank to view default Modbus objects for a device. Refer to *Application Note* for other types available	DT	Obj\Text: 20 chars; 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:

```
real-value = (M x raw-value) + A
```

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

```
raw-value = (real-value - A) / 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:

temperature = (0.0015259 x 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 serial devices available.

Description	Reference	Type
Address *x* The Modbus device address number, *x,* is in the range 1…247	A*x*	Fixed container, one of the following: Default Modbus device *[Modbus\Default]* If the Device Type is configured then the container will be of the type [Modbus*Device Type*]. Refer to *Application Note* for more information

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 the *Understanding Modbus* section 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
Coil *r* - State The digital output, *r*, is in the range 0…65535 (see note 1)	C*r*.A	Obj\OffOn; Adjustable
Discrete Input *r* - State The digital input, *r*, is in the range 0…65535 (see note 1)	A*r*.A	Obj\OffOn
Holding Register *r* - Unsigned 16-bit Integer The register, *r*, is in the range 0…65535 (see note 1).	D*r*.B	Obj\Num; Range: 0…65535; Adjustable
Holding Register *r* - Unsigned 16-bit (x10)	D*r*.B26	Obj\Float; Range: 0…6553.5; Adjustable
Holding Register *r* - Unsigned 16-bit (x100)	D*r*.B27	Obj\Float; Range: 0…655.35; Adjustable
Holding Register *r* - Unsigned 16-bit (x1000)	D*r*.B28	Obj\Float; Range: 0…65.535; Adjustable
Holding Register *r* - Signed 16-bit Integer	D*r*.C	Obj\Num; Range: -32768…32767; Adjustable
Holding Register *r* - Signed 16-bit (x10)	D*r*.C26	Obj\Float; Range: -3276.8…3276.7; Adjustable
Holding Register *r* - Signed 16-bit (x100)	D*r*.C27	Obj\Float; Range: -327.68…327.67; Adjustable
Holding Register *r* - Signed 16-bit (x1000)	D*r*.C28	Obj\Float; Range: -32.768…32.767; Adjustable
Holding Register *r* - IEEE Float Stored in two registers (MSW, LSW)	F*r*.J	Obj\Float; Adjustable
Holding Register *r* - Unsigned 32-bit Integer Stored in two registers (see note 2)	F*r*.B	Obj\Num; Range: 0…4294967295; Adjustable
Holding Register *r* - Signed 32-bit Integer Stored in two registers (see note 2)	F*r*.C	Obj\Num; Range-2147483648…2147483647; Adjustable
Holding Register *r* – Unsigned 64-bit Integer Stored in four registers (see note 2)	H*r*.B	Obj\Text; Range: 0…18446744073709551615; Adjustable
Holding Register *r* – Signed 64-bit Integer Stored in four registers (see note 2)	H*r*.C	Obj\Text; Range: -923372036854775808… 923372036854775807; Adjustable
Holding Register *r* – IEEE Double Float Double precision, stored in four registers (MSW… LSW) (see note 2)	H*r*.J	Obj\Text; Adjustable
Holding Register *r* - Bit *b* The bit number, *b*, can be in the range 0…15. Refer to *Single Bit of Register* section earlier in this manual	D*r*.K*b*	Obj\OffOn; Adjustable

Description	Reference	Type
Input Register *r* - Unsigned 16-bit Integer The register, *r*, is in the range 0…65535 (see note 1)	Br.B	Obj\Num; Range: 0…65535
Input Register *r* - Unsigned 16-bit (x10)	Br.B26	Obj\Float: Range: 0…6553.5
Input Register *r* - Unsigned 16-bit (x100)	Br.B27	Obj\Float; Range: 0…655.35
Input Register *r* - Unsigned 16-bit (x1000)	Br.B28	Obj\Float; Range: 0…65.535
Input Register *r* - Signed 16-bit Integer	Br.C	Obj\Num; Range: -32768…32767
Input Register *r* - Signed 16-bit (x10)	Br.C26	Obj\Float: Range: -3276.8…3276.7
Input Register *r* - Signed 16-bit (x100)	Br.C27	Obj\Float: Range: -327.68…327.67
Input Register *r* - Signed 16-bit (x1000)	Br.C28	Obj\Num; Range: -32.768…32.767
Input Register *r* - IEEE Float Stored in two registers (MSW,LSW)	Nr.J	Obj\Float
Input Register *r* - Unsigned 32-bit Integer Stored in two registers (see note 2)	Nr.B	Obj\Num; Range: 0…4294967295
Input Register *r* - Signed 32-bit Integer Stored in two registers (see note 2)	Nr.C	Obj\Num; Range: -2147483648…2147483647
Input Register *r* – Unsigned 64-bit Integer Stored in four registers (see note 2)	Pr.B	Obj\Text; Range: 0…18446744073709551615
Input Register *r* – Signed 64-bit Integer Stored in four registers (see note 2)	Pr.C	Obj\Text; Range: -923372036854775808… 923372036854775807
Input Register *r* – IEEE Double Float Double precision, stored in four registers (MSW… LSW) (see note 2)	Pr.J	Obj\Text
Input Register *r* - Bit *b* The bit number, *b*, can be in the range 0…15. Refer to *Single Bit of Register* section earlier in this manual	Br.K*b*	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. eg 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
Function *f*, Entry *r* - Decode *d* The function, f, is in the range A…T (see note 3) The entry, *r*, is in the range 0…65535 (see note 1) The decode, *d*, is in the range A…M (see note 4)	*fr.d*	The value type is dependent on the function, f, and decode, d. Adjustable for Coils and Holding Registers
Function *f*, Entry *r* - Decode *d*, Formula *z* As above, but with formula, *z*, applied (see note 2)	*fr.dz*	The value type is dependent on the function, f, decode, d, and formula, z. Adjustable for Coils and Holding Registers
Function *f*, Entry *r* - Bit *b* The function, *f*, is in the range B,D…T (see note 3) The entry, *r*, is in the range 0…65535 (see note 1) The bit number, *b*, can be in the range 0…15. Refer to Single Bit of Register section earlier in this manual	*fr*.K*b*	Obj\OffOn; Adjustable for Coils and Holding Registers
Function *f*, Entry *r* – Bit Mask *m* The function, *f*, is in the range B,D…T (see note 3) The entry, *r*, is in the range 0…65535 (see note 1) The bit mask, *m*, is a number in the range 0…65535. Refer to Bit Mask section earlier in this manual	*fr*.L*m*	Obj\Num
Function *f*, Entry *r* – Bit Mask *m*, Formula *z* As above, but with formula, *z*, applied (see note 2)	*fr*.L*m.z*	Obj\Float

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
21	10	0
22	100	0
23	1000	0
24	10000	0
25	100000	0
26	0.1	0
27	0.01	0
28	0.001	0
29	0.0001	0
30	0.00001	0

Formula	Multiply	Add
31	2	0
32	5	0
33	0.2	0
34	0.5	0
35	0.05	0
36	0.005	0
37	0.000001	0
38	1	0
39	1	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…T. 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 5 output multi-registers	03	16
J	Holding Registers	Read/Write 6 output multi-registers	03	16
K	Holding Registers	Read/Write 7 output multi-registers	03	16
L	Holding Registers	Read/Write 8 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 5 input multi-registers	04	
R	Input Registers	Read 6 input multi-registers	04	
S	Input Registers	Read 7 input multi-registers	04	
T	Input Registers	Read 8 input multi-registers	04	

4. The decode, *d*, is in the range A…M. 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
K	Single bit of register	Obj\OffOn
L	Bit Mask	Obj\Num
M	IEEE Float (LSW, MSW order)	Obj\Float

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'.
6. All objects may be prefixed with W*g*, X*g*, Y*g*, or Z*g* to group objects together. This is useful when creating object contents files. The group number, *g*, is simply added to the discrete input, coil, or register number, *r*.
   For example, object 'X10.A1.A' and object 'A11.A' will both read the same discrete input 11.

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)

Next Steps…

If you require help, contact support on 01273 694422 or visit *www.northbt.com/support*

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

Author: JF
Checked by: BS

Document issued 18/06/2018.