Test tool for ModBus devices

Overview

This tool allows to send values directly to a ModBus slave to test if communication settings are valid, if the address or the type of a ModBus register works as expected and explained in the documentation of the manufacturer. No configuration in application or in DGQG is required to use this tool. The DINTMB02 or and DNET02 just have to be scanned and added in application to be able to use this tool.

Specifications/Limitations

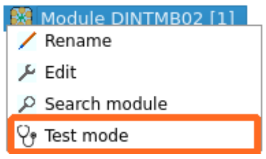
- **GoldenGate version 20.3.0** or higher is required.
- A NewGen DGQG is required (DGQG02, DGQG04, ...).
- A **DINTMB02** is requires for ModBus RTU.
- A **DNET02** is required for ModBus TCP (not yet implemented).
- The DINTMB02 interface module can only read up to five 16-bit registers at once.

Important note

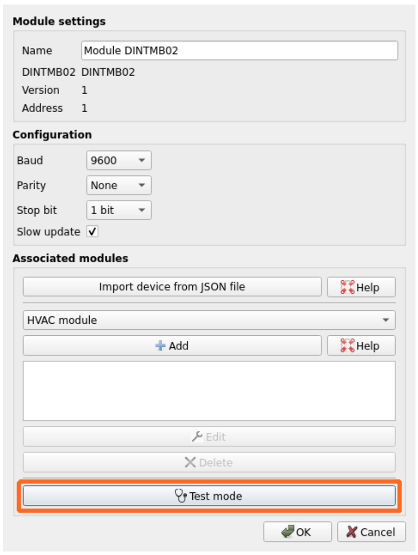
No free assistance will be provided by the Domintell support for the configuration and/or the integration of a ModBus slave. A paid support is however available if you need some assistance.

How to access

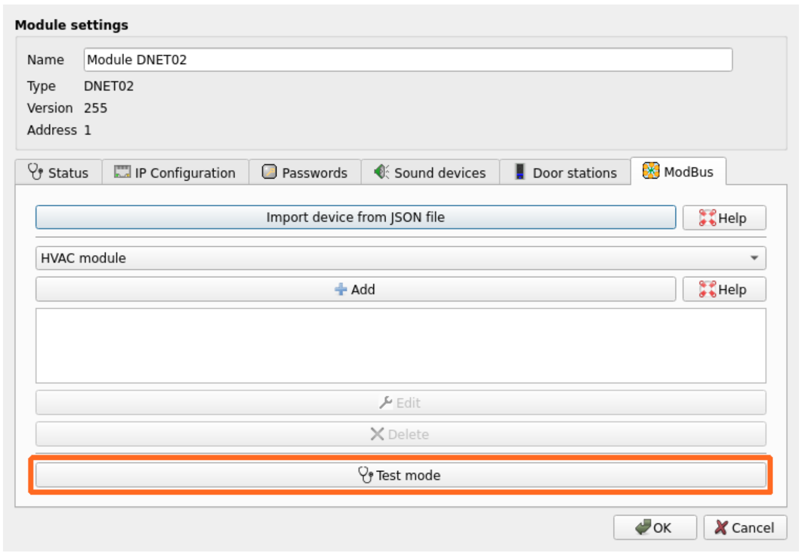
- An application containing a DINTMB02 or a DNET02 is required and must be connected to the Domintell installation.
- GoldenGate must be connected to DGQG.
- The test dialog can be opened by several ways:
    - Right-click on the DINTMB02 interface module in the output interface node in the right column of the main window.



- Using the "Test mode" button in the configuration dialog of the interface module (DINTMB02 or DNET02).

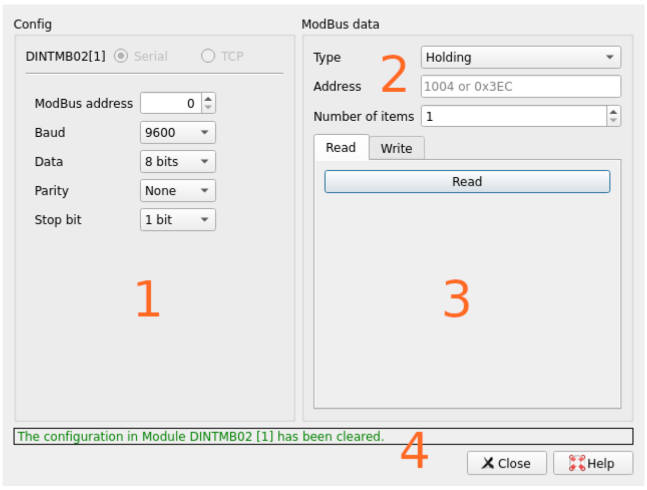


Name: Module DINTMB02
DINTMB02: DINTMB02
Version: 1
Address: 1
Baud: 9600
Parity: None
Stop bit: 1 bit
Slow update: ✓



Name: Module DNET02
Type: DNET02
Version: 255
Address: 1

- Once opened the test dialog will disable polling of ModBus devices by the interface module (DINTMB02 or DNET02). It is divided in 4 zones:



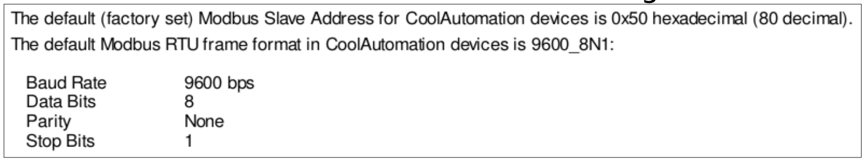
DINTMB02[1] ● Serial ○ TCP
ModBus address: 0
Baud: 9600
Data: 8 bits
Parity: None
Stop bit: 1 bit
Type: Holding
Address: 1004 or 0x3EC
Number of items: 1

The configuration in Module DINTMB02 [1] has been cleared.

  - 1 : Communication settings and the slave ID of the ModBus slave.
  - 2 : Settings of the ModBus register to read/write.
    - Holding registers and Coils can be read and written.
    - Input registers and Discrete inputs can only be read.
  - 3 : Data read or to be written.
  - 4 : The Status bar containing information about the last transaction.

Settings of communication (zone 1)

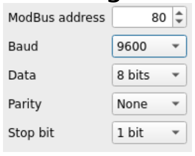
The CoolMaster ModBus device (from coolautomation.com) will be used as example. Its datasheet gives information about its slave ID and serial configuration.



The default (factory set) Modbus Slave Address for CoolAutomation devices is 0x50 hexadecimal (80 decimal).
The default Modbus RTU frame format in CoolAutomation devices is 9600_8N1:

Baud Rate	9600 bps
Data Bits	8
Parity	None
Stop Bits	1

The test dialog will be configured as the following:



ModBus address	80
Baud	9600
Data	8 bits
Parity	None
Stop bit	1 bit

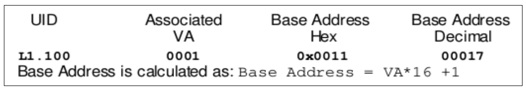
Settings of ModBus register (zone 2)

- *Type* : the type of the register to read/write
    - Holding (used function 0x03, 0x06 and 0x10)
    - Coil (used function 0x01, 0x05 and 0x0F)
    - Input (used function 0x02)
    - Discrete input (used function 0x04)
- *Address* : the address of the ModBus register to read/write. Domintell uses 0-based addresses (aka ModBus address). Depending of the manufacturer, addresses of registers in the documentation can be 0-based (aka ModBus address) or 1-based (aka Modicon/PLC address).

  If the datasheet of the ModBus slave gives address in 1-based format, e.g. 203, the address 202 must be used anywhere in Domintell ecosystem.
- *Number of items* : number of 16-bit registers (or holding/input registers) or number if bits (for coils/discrete inputs) to read/write from the specified address.

Read Data (zone 3)

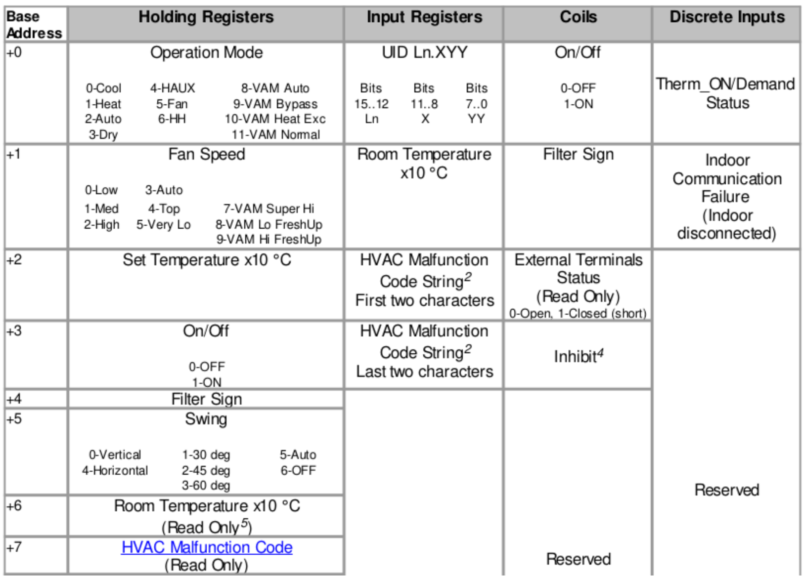
The base address of ModBus registers of the CoolMaster salve is 17 (1-based). The base address to use in Domintell will be 16.



UID	Associated VA	Base Address Hex	Base Address Decimal
L1.100	0001	0x0011	00017

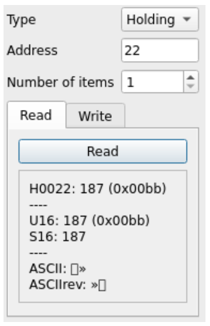
Base Address is calculated as: `Base Address = VA*16 +1`

Here is the list of available registers :



Base Address	Holding Registers	Input Registers	Coils	Discrete Inputs
+0	Operation Mode 0-Cool 1-Heat 2-Auto 3-Dry 4-HAUX 5-Fan 6-HH 8-VAM Auto 9-VAM Bypass 10-VAM Heat Exc 11-VAM Normal	UID Ln.XYY Bits 15..12 Ln Bits 11..8 X Bits 7..0 YY	On/Off 0-OFF 1-ON	Therm_ON/Demand Status
+1	Fan Speed 0-Low 1-Med 2-High 3-Auto 4-Top 5-Very Lo 7-VAM Super Hi 8-VAM Lo FreshUp 9-VAM Hi FreshUp	Room Temperature x10 °C	Filter Sign	Indoor Communication Failure (Indoor disconnected)
+2	Set Temperature x10 °C	HVAC Malfunction Code String[2] First two characters	External Terminals Status (Read Only) 0-Open, 1-Closed (short)	Reserved
+3	On/Off 0-OFF 1-ON	HVAC Malfunction Code String[2] Last two characters	Inhibit[4]	
+4	Filter Sign		Reserved	
+5	Swing 0-Vertical 4-Horizontal 1-30 deg 2-45 deg 3-60 deg 5-Auto 6-OFF			
+6	Room Temperature x10 °C (Read Only[5])			
+7	HVAC Malfunction Code (Read Only)			

- Read **one register of 16 bits**



```
H0022: 187 (0x00bb)
----
U16: 187 (0x00bb)
S16: 187
----
ASCII: □»
ASCIIrev: »□
```

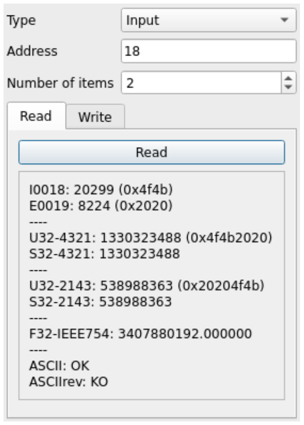
Room temperature @ H0022

Several lines are displayed as reply :

- ○ H0022 : contains raw data in decimal and in hexadecimal format.
- ○ U16 : contains data interpreted as an unsigned 16-bit word (will contain same data as first line).
- ○ S16 : contains data interpreted as a signed 16-bit word (two's complement).
- ○ ASCII : contains data interpreted as printable character string. The highest byte of the 16-bit data will be the first character and the lowest byte will be the second character. If 16-bit data is 0x3132, "12" will be displayed.
- ○ ASCIIrev : same as ASCII except that the lowest byte of 16-bit data is the first character and the highest byte is the second character. If 16-bit data is 0x3132, "21" will be displayed.

From the datasheet, the temperature in the room is 187 * 0.1°C = 18.7°C

- Read **two registers of 16 bits** (**32-bit register**)



```
I0018: 20299 (0x4f4b)
E0019: 8224 (0x2020)
----
U32-4321: 1330323488 (0x4f4b2020)
S32-4321: 1330323488
----
U32-2143: 538988363 (0x20204f4b)
S32-2143: 538988363
----
F32-IEEE754: 3407880192.000000
----
ASCII: OK
ASCIIrev: KO
```

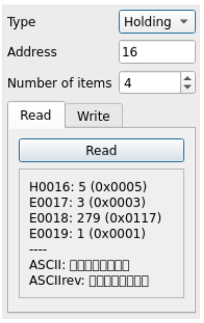
H.V.A.C. malfunction code string @ I0018

Several lines are displayed as reply :

- ○ I0018 : contains first raw 16-bit data in decimal and in hexadecimal format.
- ○ E0019 : contains next raw 16-bit data in decimal and hexadecimal format ('E' means "Extra data" of I0018, so E0019 = I0019).
- ○ U32-4321 : contains data interpreted as an unsigned 32-bit word in decimal and hexadecimal format with 16-bit data of I0018 placed at highest bits and 16-bit data of I0019 at lowest bits. This is the most common data format for unsigned 32-bit data words used by ModBus slaves.

   - S32-4321 : same as U32-4321 except that data is interpreted as an signed 32-bit word (two's complement). This is the most common data format for signed 32-bit data words used by ModBus slaves.
   - U32-2143 : same as U32-4321 except that 16-bit data of I0018 is placed at lowest bits and 16-bit data of I0019 is placed at highest bits of the 32-bit word.
   - S32-2143 : same as U32-2143 except that data is interpreted as an signed 32-bit word (two's complement).
   - F32- IEEE754 : contains data interpreted as a IEEE 754 floating-point decimal value. The U32-4321 data format with scale factor (x10, x100, ...) is more common than the F32-IEEE754 format.
   - ASCII : contains data interpreted as printable character string. The highest byte of the 16-bit data will be the first character and the lowest byte will be the second character. If 16-bit data is 0x3132, "12" will be displayed. The next 16-bit data is decoded and append in the same way.
   - ASCIIrev : same as ASCII except that the lowest byte of 16-bit data is the first character and the highest byte is the second character. If 16-bit data is 0x3132, "21" will be displayed. The next 16-bit data is decoded and append in the same way. In other words, ASCIIrev is not just the full ASCII string printed in the opposite direction!

- Read **three or more registers of 16 bits**



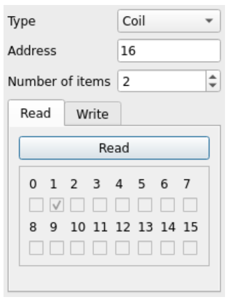
Type	Holding
Address	16
Number of items	4

```
H0016: 5 (0x0005)
E0017: 3 (0x0003)
E0018: 279 (0x0117)
E0019: 1 (0x0001)
----
ASCII: □□□□□□□□
ASCIIrev: □□□□□□□□
```

Read H0016 to H0019

When reading more than two 16-bit registers, only individual registers and ASCII representation are displayed.

- Read **1-bit** register



Type	Coil
Address	16
Number of items	2

0	1	2	3	4	5	6	7
☐	☑	☐	☐	☐	☐	☐	☐
8	9	10	11	12	13	14	15
☐	☐	☐	☐	☐	☐	☐	☐

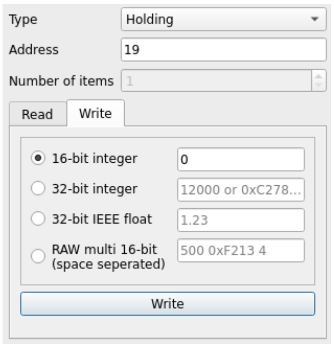
Read C0016 and C0017

Two coils registers have been read. C0016 is at offset 0 and C0017 is at offset 1. C0016 is unchecked so value is 0. Coolmaster is therefor "off". C0017 is checked so value is 1. "Filter sign" is true.

Write data (zone 3)

When writing data, number of data to write is automatically defined by from the type of chosen register.

- Write **one register of 16 bits**



Type: Holding
Address: 19
Number of items: 1
Read | Write
(●) 16-bit integer: 0
( ) 32-bit integer: 12000 or 0xC278...
( ) 32-bit IEEE float: 1.23
( ) RAW multi 16-bit (space seperated): 500 0xF213 4
Write

Write 0 to H0019

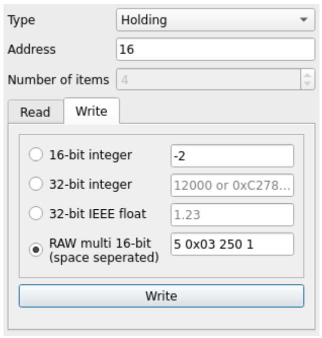
The value (from 0 to 65 535 or from -32 768 to 32 767) can be expressed in decimal format or in hexadecimal format (prefixed with "0x").

- Write **one register of 32 bits**

  This is the same as 16-bit mode but for this mode, the function 0x10 (Write Multiple Registers) is used to write two 16-bit register at once. The value can vary from 0 to 4 294 967 295 or from -2 147 483 648 to 2 147 483 647.

- Write **several registers of 16 bits**

  Values must be seperated by a space and can be either in decimal format or hexadecimal format.



Type: Holding
Address: 16
Number of items: 4
Read | Write
( ) 16-bit integer: -2
( ) 32-bit integer: 12000 or 0xC278...
( ) 32-bit IEEE float: 1.23
(●) RAW multi 16-bit (space seperated): 5 0x03 250 1
Write

Write 4 values starting from H0016

CoolMaster will be set to fan mode, auto speed, set-point to 25°C and is enabled.

Status bar (zone 4)

Several messages can be displayed in status bar.
Three colors are used:

- Blue : operation in progress
- Green : operation successfully ends
- Red : operation ends with errors.

Following messages can be expected :

- "The configuration in <module> has been cleared." : the current configuration in the interface module has been cleared, the polling (continuous interrogation) of ModBus slaves is stopped and any request by the current application in the DGQG will be rejected. The ModBus bus is fully dedicated to the test dialog of GoldenGate.

- "Original configuration has been loaded." : the original configuration has been restored in the interface module, the application in DGQG can read/write ModBus registers and ModBus slaves are polled again and changes are notified to DGQG.
- "Register Tnnnn read" : the content of ModBus register has been read, decoded and displayed (in zone 3)
- "Data written to Tnnnn..." : data have been successfully written to the ModBus slave.
- "The command was rejected... (NACK-0xnn)" : the command was rejected by the interface module because it did not have the correct format. Check if the interface module runs the last firmware version. If it has the last firmware version, please contact the support of Domintell with the code given after NACK.
- "Malformed frame received. Please check the version of <module>!" : The frame received from the interface module (DINTMB02 or DNET02) is malformed. Check if the interface module runs the last firmware version or you us the last version of GoldenGate.
- "Unable to preform write operation!" : the write operation could not complete correctly.
- "No data to display!" : an empty set of data has been received. There is nothing to decode and display.
- "Unable to perform read operation! (nn-ccccc)" : the ModBus slave rejects the operation. 'nn' (in hexadecimal) is the ModBus error/exception code and 'ccccc' is the human readable name of the error/exception.
    - 01 = ILLEGAL FUNCTION : the ModBus function/command is not handled by the ModBus slave. e.g. the ModBus slave does not support the 0x0F function (Write Multiple Coils).
    - 02 = ILLEGAL DATA ADDRESS : the address of the register does not exist in the ModBus slave or the last address of the register to be written does not exist (too much data to be written).
    - 03 = ILLEGAL DATA VALUE : the value to be written is out of range.
    - 04 = SERVER DEVICE FAILURE : the ModBus slave experiences some issues.
    - 05 = ACKNOWLEDGE : the ModBus slave accept the frame but it will delay its execution
    - 06 = SERVER DEVICE BUSY : the ModBus slave is busy and do not accept any frame.
    - 08 = MEMORY PARITY ERROR : the ModBus slave has some memory integrity's issues.
    - 0A = GATEWAY PATH UNAVAILABLE : the ModBus slave is a ModBus gateway and the path where the frame must be forwarded can not be defined. e.g. the ModBus slave forwards a RTU frame to a ModBus TCP slave but the ModBus TCP is not correctly configured in the ModBus gateway.
    - 0B = GATEWAY TARGET DEVICE FAILED TO RESPOND : the ModBus slave is a ModBus gateway and the destination ModBus slave is not connected or reachable by the ModBus gateway.
    - E0 = NO REPLY (Domintell exception) : no ModBus slave answers to the specified slave ID.
    - E1 = OUT OF REGS (Domintell exception) : the number of 16-bit registers to be written/read is too high for the interface module. The interface module can normally read five 16-bit registers at once.
    - E2 = BUILD FRAME ERROR (Domintell exception) : the interface module can not generate the ModBus frame.

- "Processing the register Tnnnn..." : data have been received from the ModBus slave for the register Tnnnn and are being decoded.
  - 'T' can be one of the following :
    - 'H' : Holding register (16 bits)
    - 'C' : Coil (1 bit)
    - 'I' : Input (16 bits)
    - 'D' : Discrete input (1 bit)
  - 'nnnn' is the address of the register in decimal format. "H40004" means holding register at address 40004 (0x9C44).
- "Register written..." : the register has been written to the ModBus interface and an answer is expected from the ModBus slave.