User Guide
Document Ref.: 8809
21_03_2022
Version: 1.1

Hera Modbus to MQTT application

Facilitates reading and writing of Modbus registers via MQTT with a Hera

Version number	Date	Author	Changes
0.9	31 October 2018	P. Tupper	Initial draft
1.0	4 February 2022	P.Tupper	Added definition of '-INPUT' to coils/registers to utilise the input register addresses
1.1	21 March 2022	J. van den Broek	Updated styles

Table of Contents

1. Introduction

1.1 Purpose

Many PLC/SCADA systems offer modbus as a mechanism to configure and read system parameters. This is possible remotely by using a modbus relay agent for RTU(serial) or TCP. However the bandwidth required on the data connection can be reduced by enabling the relay agent within the router (Hera) to read/write register values and use json to convey the get/set parameters to a remote service. MQTT is an ideal transport mechanism for the register data and its use enables connection to numerous cloud services (AWSIoT, Microsoft Azure etc.) or private broker.

This application enables autonomous polling of registers or groups of registers with configured intervals and generates json reports to be sent to a remote service over MQTT. It is also possible to poll register groups 'manually' and to write registers via remote MQTT commands.

Parameters read from a remote device can be continually updated on each poll interval or each parameter group can be configured to only send updates when a value within the group changes. This allows for reduced data usage but retains real-time status updates.

1.2 References

None

2. Technical Summary

Each group of registers (a group can contain a single register if required) is read according to a specified interval and parameters collated into a report formatted in json to publish to the MQTT broker. If configured to report on changes to the group all values are stored in volatile memory and upon each poll the data is compared to that previously read. If there are changes to any register/coil within the group the entire group report is published to the broker. A maximum time between reports can also be set for a group to ensure a report is periodically published even without any parameter changes.

All coils/registers of which the application needs to be aware (including write-only parameters) must be declared within the register configuration file. The application does not use the full register addresses to determine whether the address refers to a coil or register - instead within the register config file it must be specified what 'type' the address refers to (BIT/REGISTER/REGISTERS/DWORD/DWORDS/FLOAT/FLOATS/STRING). For REGISTERS, DWORDS, FLOATS or STRING the number of chained registers or length of string must also be specified.

Groups are not used when writing registers but can be specified to help uniquely identify a register. If multiple registers are defined with identical names but different groups they can be included within individual groups where the group name becomes a prefix to the register name. These groups need not be configured to poll but allow for multiple identical register names to be used.

If a request is received to write a register without a group all registers found in all groups with a matching name will be written.

3. Usage and Configuration

3.1 Application configuration

Configuration for the application is contained in /etc/config/modbus_mqtt. This file contains parameters to configure the MQTT and modbus connections. Modbus registers and polling are defined in the modbus configuration file which is separate from this file.

etc/config/modbus_mqtt file:

config modbus_mqtt modbus_mqtt

option disabled '1' – set 1 to disable this application

option use_certs_from_sim 'no' – used in conjunction with anynet secure SIM to read security details from an installed SIM.

option configled 'WiFi' – the LED to use to show that anynet secure credentials have been read

option connectedled 'Wan' – the LED to indicate successful MQTT connection

option privatekey '/var/private_key' – storage location for SSL private key

option clientcert '/var/public_cert' – storage location for SSL public certificate

option rootca '/var/rootca' – storage location for SSL root CA

option usetls '1' – Use SSL connection for MQTT

option clientid 'modbus_mqtt' – client ID for MQTT broker

option mqtturl '192.168.1.133' – URL for MQTT broker

option publishtopic 'modbus_mqtt_report' – mqtt topic to publish reports to

option directivetopic 'modbus_mqtt' – MQTT topic to subscribe to listen for commands

option qos 0 – MQTT qos

option mqttkeepalive 60 – MQTT keepalive

option tracefile '/var/logger/modbus_mqtt' – file to trace debug messages to

option uartdevice '/dev/eser0' – uart for modbus RTU

option uartstopbits '1' – uart stop bits for modbus RTU

option uartdatabits '8' – uart data bits for modbus RTU

option uartparity '0' – uart parity for modbus RTU (O/E/N)

option uartbaud '9600' – uard baud for modbus RTU

option modbus_tcpport '502' – modbus TCP port number

option modbus_tcpurl '192.168.1.39' – modbus TCP url

option modbus_mode 'tcp' – modbus mode (tcp/rtu)

option modbus_regfile '/etc/mbconfig' – location of the modbus registers config file

option modbus_serverid '1' – slave ID for modbus

option modbus_msgcode 'report' – add a ‘code’ json field to the report with this value

3.2 Modbus operation

Each modbus ‘parameter’ must be defined individually in the modbus register file. A parameter may be a single coil (BIT)/register or a sequence of registers making up a an array or single value represented by >16 data bits. Parameter types supported from within the input/holding registers address range are:

REGISTER (16-bit integer value)

DWORD (2 x 16-bit registers combined to form a 32-bit value)

FLOAT (2 x 16-bit registers combined to form 32-bit value representing a single-precision floating point number)

input/holding register addresses can be sequenced together to form:

REGISTERS (an array of 16-bit integer values read sequentially from a base address)

DWORDS (an array of DWORD elements read sequentially from a base-address)

FLOATS (an array of FLOAT elements read sequentially from a base-address)

STRING (an array of 16-bit integer registers read as 8-bit characters (2 per register))

In terms of modbus communication registers are read in the same way for REGISTER(S)/DWORD(S)/FLOAT(S)/STRING but the resulting json report is collated differently.

3.2.1 Addressing

Modbus addressing allows for access to coil outputs, digital inputs, analogue inputs and holding registers depending on the parameter configuration.

BITs are read using function code 1 and written with function code 5. BITs marked as input (base address 10001) are read using function code 2 and cannot be written.

REGISTERs (base address 40001) are read using function code 3 and written with function code 16. REGISTERs marked as input (base address 30001) are read using function code 4 and cannot be written.

3.3 Modbus configuration

Configuration for modbus registers and polling is contained in the modbus register file specified in the /etc/config/modbus_mqtt as 'modbus_regfile'. This file allows configuration of register groups (used for polling) and individual registers.

A group definition looks like: '$groupdef, <groupname>, <mqttretainflag>, <pollperiod>[, <reportonchange>[, <maxreportinterval>]]'

e.g. $groupdef, TEST1, 20, true, 120

$groupdef is an indicator that this line is a group definition.

groupname is a unique name for a group of registers and is used to generate reports

mqttretainflag is a boolean (true/false/1/0) that specifies whether the retain flag should be set on the mqtt publish for the json report.

pollperiod is the time in seconds between polls of the groups registers

reportonchange is true/false/1/0 to request generation of group reports only if one of the parameter values has changed since the last report

maxreportinterval forces a group report to be generated (even if no parameters have changed) if this period of time has elapsed since the last report for this group.

A register definition looks like: '<groupname>, <registername>, <registertype>, <registeraddress>[, <polltime>'.

e.g. TEST1, REG1, REGISTER, 1, 0

groupname is the name for the register group to which this register belongs it should match a group defined by a $groupdef definition although if it does not a new group with the groupname will be created.

registername is a name used in the generated report and also used when issuing a set command.

Registertype defines the type of register to be read. Options are: BIT, REGISTER, REGISTERS, DWORD, DWORDS, FLOAT, FLOATS, STRING. REGISTERS, DWORDS, FLOATS and STRING MUST be suffixed with the number of elements. e.g. DWORDS10 will read 10 double-word registers. STRING20 will read a 20 character string. registertype can also specify byte ordering and register ordering (for multi-register reads/writes). Suffixing with '-BS' swaps the bytes for each register and '-RR' reverses the order of the registers. Suffixes -HEX and -FLOAT can also be used as report-modifiers. A -HEX suffix on any of the register types results in the json report showing the values in hexadecimal. A -FLOAT suffix on DWORD/DWORDS will convert the report value to a float – this is the same as using FLOAT/FLOATS as the register type. Suffixing -HEX on a FLOAT/FLOATS will report the value as HEX – this is the same as using DWORD/DWORDS with the -HEX suffix.

Coils and registers can be set as inputs by adding the suffix '-INPUT' to the registertype.

Registeraddress is the offset address from the base of the register type. For BITs (coils) addresses are offsets from 00001. Input BITS are offset from 10001. Input REGISTERs are offset from 30001 and other registers are offset from the holding registers base (40001).

polltime is included for backward compatibility and need not be specified if the group has a definition or another register in the same group has a polltime set. If there is a group definition this polltime is ignored. If there is no group definition the LOWEST value of polltime for any of the registers in the same group is used for the group poll time (except for 0).

3.4 Operation

On boot the application attempts to connect to the mqtt broker using the configured parameters. If 'use_tls' is set it will use SSL for the connection.

Registers are read from the modbus config file and sorted into their groups. The group poll interval is determined from the group definition or if no definition is present from the polltimes defined for each register within the group. If the group is defined explicitly with $groupdef the polltime value overrides those specified for the individual registers. If there is no $groupdef the smallest value polltime for any register within the group (except 0) is used for the group polltime. If a group poll time is 0 the group will never poll on a schedule but can be polled manually by mqtt command.

Defining a group with no poll time is required to enable registers to be settable via mqtt without polling.

3.4.1 Report only on change

If reportonchange is set for a register group they will still be polled according to the group polltime. If subsequent reads show all register values to be identical to those read previously no report will be generated. This feature can be useful to save network bandwidth where knowledge of register

changes is required but periodic polled reports are not. The maxreportinterval value is used to force a report to be generated periodically if reportonchange is set but no register values have changed.

3.4.2 Report format

Reports of group parameters are presented in json format. Multiple parameter values (REGISTERS/DWORDS/FLOATS) are not concatenated and are presented as an array of elements of the defined type. i.e. a register definition for TESTREG of type REGISTERS3 generates a json element in the report of “TESTREG1”:<value1>,”TESTREG2”:<value2>,”TESTREG3”,<value3>. However STRING types are presented as an ASCII string to the NULL terminator (if one is present in the registers read). i.e. a register definition for MYSTRING of type STRING10 generates a json element in the report of “MYSTRING”:”mystring” and does not encode the trailing NULL.

Json values for BIT parameters are true/false (json boolean). Values for regular REGISTER(S) and DWORD(S) are integer numbers (unquoted json integer). Values for hexadecimal, float and string reports are reported as json strings (quoted).

The complete report format for a register group consists of:

{“code”:”<groupname>”,”values”:{“<reg1>”:”<reg1value>”,”<reg2>”:”<reg2value>”}}

3.4.3 Writing registers

Registers can be written by publishing a json-encoded mqtt message to the directivetopic with the ‘code’ value ‘settings’. BIT values can be set with ‘true’ or ‘false’. STRING values are written with a json string. Integer numeric values (REGISTER/REGISTERS/DWORD/DWORDS) are written with json int values unless the register definition has the -HEX suffix or the json value begins with’0x’ in which case the value is read as hexadecimal. FLOAT(S) parameters are assumed to be float strings unless the value begins with ‘0x’ in which case it will be read as hexadecimal. As with polling multi-register fields must be written as individual values eg:

TESTREG REGISTERS3 is written as {“code”:”settings”,”values”:{“TESTREG1”: 65535, “TESTREG2”: 65535, “TESTREG3”:65535}

OR

{“code”:”settings”,”values”:{“TESTREG1”: “0xFFFF”, “TESTREG2”: “0xFFFF”, “TESTREG3”:”0xFFFF”}

TESTDWORD DWORDS2 is written as {“code”:”settings”,”values”:{“TESTDWORD1”:4294967295, “TESTDWORD2”:4294967295}

OR

{“code”:”settings”,”values”:{“TESTDWORD1”:”0xFFFFFFFF”, “TESTDWORD2”:”0xFFFFFFFF”}

TESTFLOAT FLOATS2 is written as {“code”: “settings”, “values”:{“TESTFLOAT1”:”1.0001”, “TESTFLOAT2”:”1.0002”}}

OR

{“code”: “settings”, “values”:{“TESTFLOAT1”:”0x1234”, “TESTFLOAT2”:”0x1234”}}

Alternatively complete REGISTERS/DWORDS/FLOATS can be written using a string containing a hex number. The json for this is:

{“code”:”settings”,”values”:{“TESTDWORD”,”0x0123456789abcdef”}}

All register values are treated as unsigned.

Registers defined with a suffix of -HEX will only be written with a hex value (with or without the leading ‘0x’).

3.4.4 Debug interface

Test commands can be sent to the modbus_mqtt application by writing to /var/modbus_mqttcmd. Currently the only supported command is ‘setreg “regname” “regvalue”’.

3.4.5 Changing register configuration

A new version of the modbus register file can be uploaded via mqtt using the ‘reg_content’ directive code. The file is included as raw ascii within the json of the directive. The values required within the json string are ‘file’ and ‘content’ where file is the full path the file should be saved to within the hera and content is the raw content of the file.

e.g. {“code”:”reg_content”,”values”:{“file”:”/etc/modbus_config_test”,”content”:”<raw content of file>”}}

Once the file has been successfully stored to the file system the value of ‘modbus_regfile’ within the modbus_mqtt application config file will be changed the match the new file path. The file will then be reloaded and polling will start immediately using the newly received modbus register configuration.

4. Hardware requirements

This application is written to run on the Hera604 and future openWRT-based Hera platforms. A serial port is required for modbusRTU and an external RS485 converter is required to interface to RS485 PLCs. Modbus TCP is presented via ethernet or WiFi.