CAREL – Confidential



**REQUIREMENTS SPECIFICATION**

*Framework for*

*Gateway Middle End*

rev. 0.2

DRAFT

Index

[Index 2](#_Toc23342428)

[Revision 3](#_Toc23342429)

[1 Introduction 4](#_Toc23342430)

[1.1 Scope of the document 4](#_Toc23342431)

[1.2 Definitions, acronyms and abbreviations 5](#_Toc23342432)

[2 Operating principle 6](#_Toc23342433)

[2.1 Initialization 6](#_Toc23342434)

[2.2 Regular operation 10](#_Toc23342435)

[2.3 File transfer on GME 10](#_Toc23342436)

[3 Implementation 12](#_Toc23342437)

[3.1 Data structures 13](#_Toc23342438)

[3.2 Initialization procedure 14](#_Toc23342439)

[3.3 Regular operation procedure 19](#_Toc23342440)

[4.1 File transfer on GME 20](#_Toc23342441)

Revision

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Rev. date | Author | Note |
| *0.1* |  | *V.Cellini* | 1st draft |
| 0.2 |  | A.Bilato | Added some note |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Introduction
   1. Scope of the document

This document refers to the realization of a software framework, implementing the so called Middle End Gateway (GME). Such device is intended for collecting data from a single connected device and for sharing such information with a MQTT broker (to whom it is connected in WiFi or 2G). In detail, GME:

* gets data from a single Modbus RTU slave connected device via RS485/TTL,
* temporarily keeps data and sends it to a MQTT broker (2G/WiFi),
* Receive requests of changes of variable values from the MQTT broker and resends them to the connected device,
* Receive a file name and url of the model info file via MQTT and download the file itself via HTTPS .
* Receive a file name and url of the new GME FW file via MQTT and download the file itself via HTTPS.

This document aims at describing the basic functioning of the GME and at identifying the building blocks of the overall firmware.

The framework originated by this document is intended to be portable on different hw platforms.

FW will be made of different portions of code, some of which will be implemented by Carel, while others will be up to USR. For example, all the features that require direct access to hardware peripherals will be implemented by USR and Carel will be able to drive them, when needed, by calling some primitives (e.g. MQTT communication primitives will be implemented by USR, but Carel will need to call some of these primitives to transmit MQTT messages with custom Carel payloads). On the other hand, logical management actions will be implemented by Carel and USR will be instructed on how to start such managements, by calling some specific functions (e.g. polling engine disciplining data gathering will be implemented by Carel and USR will just need to be sure that the proper initialization function is called).

Integrating and building sources is up to USR.

To this extent, Carel will provide documentation describing the functions to call to implement different functions, whereas USR should provide Carel a set of primitives (i.e. header files) to drive peripherals, manage Modbus and MQTT messaging, etc.

To ease this cooperation work, in some cases, Carel will provide not fully implemented functions, to be filled in by USR. More details on FW organization will be given in Implementation section.

This document is organized as follows. At first, the operating principle is described, focusing on actions to be carried on during initialization and during regular operation. Then, more detail on implementation is given.

* 1. Definitions, acronyms and abbreviations

AP = WiFi access point to connect to

GME\_AP = gateway that act as an access point  
GME\_WIFI = Gateway Middle End versione WiFi

GME = Gateway Middle End  
GSM = refer to 2G/4G/NB IoT connection  
FW = firmware

IoT = Internet of Things

MonDev = the device connected to the GME through the RS485 interface

OTA = Over The Air

SW = software

“model file” = a binary file that contains the data needed by the GME to read and write via ModBus the connected device.

MODEL\_TABLE = table defining the meaning of Modbus registers build from “model file”

GTW000MWT0 = the GME-WiFi model

GTW000MGP0 = the GME-2G model with CAREL SIM installed

GTW000MGT0 = the GME-2G model without SIM

1. Operating principle
   1. Initialization

Initialization of the gateway requires the following steps:

1. Initialization of operating system (if present)
2. Initialization and check of the file system
3. Recovery of configuration/model information
4. Connection to wifi access point/2G provider
5. Initialization of the RTC
6. Initialization of the RS485
7. Initialization of MQTT engine
8. Inizialization fo the polling engine

### Initialization of operating system

We assume that a real-time operating system is available. This way, running single tasks concurrently becomes easier and more efficient.

### Initialization of file system

We assume that a file system is available and that information can be stored/recovered using some primitive functions.   
There aren't any special requirements for the file system, no directory management or in other word only the root are enough to store no more than 10 files (currently 4 files are   
foreseen).

We need to load in this file system at production time some files, take care of this.

### Initialization of button/leds

Gpio pins corresponding to button and leds must be initialized.

At this step, a task monitoring the status of the button must start.  
The only button present in the GME is the reset button, with the following behaviour.

if during regular operation it is pressed for a short time (more than 2s, less than 5s), system must be rebooted.

If, at power up, it is pressed for more than 5s, GME is reverted to its factory default status: the installed model is deleted and connection parameters are reset to default.

As regards leds, WiFi model has 2 leds, a green one for power indication and a red one for connection indication. Red led must be switched on as soon as device is powered. Green led:

1. is off if not connected to an AP.
2. blinks 1 time/s if connected to an AP but not to the CAREL MQTT server
3. is always on if connected to CAREL MQTT server.

This management will be done by CAREL, by calling a   
void Led\_Status(C\_BYTE which\_led, C\_BYTE status\_value);

On 2G model, there are five leds, one for power indication (red) and four (green) for connection indication. Red led must be switched on as soon as device is powered.

Connection leds perform (if feasible) these indications (we will refer to the current led name):

1. (WORK) blinks when GME receives data via RS485  
   This is useful to detect communication trouble
2. (GPRS) is on if the connection with the GSM provider was successfully performed
3. (LINKA) is on if the connection with CAREL MQTT server was successfully performed
4. (LINKB) shows the status of the configuration (i.e. model file present).

This management will be done by CAREL, by calling a   
void Led\_Status(C\_BYTE which\_led, C\_BYTE status\_value);

### Recovery of configuration/model information

During this phase, it must be checked that a valid configuration file is available in file system. File must be checked and read. If such file is not present or if it’s invalid, a default configuration will be considered. These actions will be managed by Carel.

Configuration file contains info about:

* baud rate of the RS485 Modbus RTU link
* Modbus address of the connected device
* url and port of the ntp server from which time will be retrieved
* url, username and password for connection to the MQTT broker
* in case of 2G gateway, the APN for mobile connection
* in case of WiFi gateway, the AP SSID and password for connection

Default configuration includes:

* baud rate 19200bps

Configuration file cannot be uploaded via OTA but can be updated via RS485 (in a way that will be detailed later).

Successively, model must be recovered the same way.  
It no such file is present or if it is not valid, a default model is considered.

Model file contains:

* the variables list that will be cyclically read from the connected device via Modbus protocol
* data format of the RS485 link (data bits, stop bits, parity)

Default model includes:

* data format 8-n-1

Model file can be updated during regular operation via OTA upload.

A file is valid if its CRC16 checksum, written at the end of the file, is valid and if it contains a well-defined header.

### Connection to wifi access point

GME must connect to a wifi access point, information about connection must be recovered from the configuration file.

At first boot, AP authentication data is not written in configuration file. So first configuration must occur. As already agreed, the WiFi model has no enough space to host an internal Webserver so, a method to configure the GME through an APP must be provided.  
A very common method used in some WiFi appliances is the one described below that requires a smartphone APP that:

1. searches the available AP and recognizes the special AP of the GME,
2. asks the user about which AP he/she wants to use to connect to internet and relative password,
3. sends this data to GME,
4. GME enables STA mode with the provided credentials and connects to the AP,
5. The APP connects to the same AP and checks for the presence of the GME.

A demo APP for Android that show the protocol used is required.

The WiFi configuration APP will be able to:

* + Select the AP name to be connected to
  + Set the AP password
  + Select the mode WPA/WPA2
  + Select a fixed IP address or use DHCP
  + Select the default gateway
  + Select the proxy name and port.

### Connection to 2G provider

There are two models of GME 2G, using the same HW, but having these differences:

1. The GTW000MGP0 mounts a CAREL SIM.  
   This model out of the box is able to connect to CAREL cloud system
2. The GTW000MGT0 doesn’t mount any SIM, the installation of the SIM is in charge to final user.

Model with Carel SIM onboard needs no configuration for connection, as APN name (bound to the SIM provider) is already written in default configuration.

In case of SIMless model, APN must be provided for connection to complete.

A possible way to accomplish this configuration may be sending an SMS to the module (using the actual SIM phone number, known to final user). Received APN and other data should be saved to file system, in configuration file, to be available after successive reboots.

To prevent unauthorized reconfiguration the SMS is protected with a password unique to each device, this password is the same out of production for all the device (ie:12345678) and changeable via SMS, so that, the 1st SMS set the new password and store it in the cloud, the 2nd SMS will configure the GME and use the new password.

Example 1st SMS – change Password

OLDPWD:12345678 NEWPWD:73216651

Example 2nd SMS – change APN

PWD:73216651 APN:……

Obviously is possible to call “SMS – change Password” every time we need not only the first time.

If you have already implemented a system like this please send us the documentation.

### Initialization of dhcp client – only for WiFi model

A dhcp client must be initialized in case GME has to work in dhcp mode, as inferred from the configuration file.

### Initialization of dns client

A DNS client must be initialized to allow GME to convert alpha-numeric names into the numeric IP addresses that are required for access to external network resources on the Internet.

### Initialization of RTC

Clock setting is performed via NTP. UTC time is obtained directly enquiring a NTP server. URL of the server and the port used for NTP connection are written in configuration file.

### Initialization of certificates

The GME must natively contain 2 identical security certificate files.

Certificate files can be updated during regular operation, one a time.  
  
The GME attempts to validate the first: if it is not valid or not validated by the server, the second is checked. The validity of certificates is checked as soon as the gateway has obtained a valid time from network.

Certificate files can be updated during regular operation.  
  
The MQTT and HTTPS connections use the same certificate.

### Initialization of TLS library

TLS 1.2 library must be supported.

### Initialization of https client

An https client must be initialized, this is used for file transfer between the cloud and the GME, ie. transfer the new security certificate.

### Initialization of MQTT connection

Connection towards a MQTT server must be initialized.

GME uses parameters in configuration file or, if they are not available, it uses default parameters.

### Initialization of RS485

The serial RS485 port must be initialized at the physical characteristics contained in model and configuration files (baud rate, data bits, stop bits, parity).

The port used for communication must be specified, differentiating between 3-way connector (RS485) or TTL port.

### Initialization of Modbus

The actions required for initializing Modbus communications must be implemented.

### Initialization of Polling engine

Polling engine is a component that disciplines how variables must be read from the connected device, in terms of timing and variable set. At initialization, buffers required to store Modbus samples must be statically allocated. No dynamic allocation will be allowed. After polling engine initialization, the mechanism is not yet started: it will be, as soon as MQTT connection is established.

* 1. Regular operation

After initialization, GME regular operation starts.

GME continuously and periodically polls the connected device via Modbus protocol to get required variable values. This mechanism is called “polling engine” and it will be completely developed by Carel. USR will expose primitives for writing/reading to/from RS485 interface, via Modbus protocol. Since it must be possible to temporarily stop polling engine, a semaphore mechanism offering this chance will be available.   
The whole variables set and corresponding sampling times are stored in model file.

Gathered data must be transferred via MQTT to a broker. USR will be responsible for such communication, whereas MQTT payload packing/unpacking will be implemented by Carel.

* 1. File transfer on GME

GME must be able to download one or more files from a remote location. Those files could be model or configuration files, certificates, FW updates for the connected device or FW updates for the GME itself.

File transfer is carried on through connection to a HTTPS server.   
Hence, GME must implement HTTPS client functionalities.

USR must share with Carel all the primitives needed to complete HTTPS file transfer.

**File transfer via HTTPS**

Anytime it is needed, a file may be recovered from the cloud using an HTTPS connection.

Whenever this occurs, file (certificate, model or FW update for the connected device) is saved to file system.

**File transfer via Modbus**

A file (specifically a FW update for the connected device) can be transferred from the GME to the connected device using the Modbus file transfer command. The file was previously obtained by the GME with an HTTPS connection and is a FW update for the connected device.  
The routine to do that is given by CAREL, the required Modbus Routine for file transfer and for HTTPS connection are in charge to USR.

**FW Upgrade**

In case GME FW is to be updated, it must be ensured that the operation is fail safe and that a recovery method of a wrong transfer is provided. This means that if for some reason an upgrade will be interrupted in the middle, after a power on/off, the system will be able, at least, to restart with the previous FW.

In any case, the possibility to upgrade the FW through RS485 or (\*)USB must be provided.   
A demo in source code “C” or Python of a Windows application that perform the serial upgrade must be provided.

(\*) the USR-GPRS-730 have a USB port is possible to use it ?

1. Implementation

To ease source code sharing between Carel and USR, Carel will provide a set of files, organized in files that can be easily identified. Files containing Carel provided source code will all have the suffix \_CAREL.\*, whereas files requiring USR action will have suffix \_IS.\*.

Files will be called after their functionality, in an intuitive manner (i.e. RTC functions will be saved in RTC\_\*.\* file).

The following table shows a summary of features and identifies who should implement different parts.

|  |  |  |  |
| --- | --- | --- | --- |
| **FEATURE** | **OPERATION** | **CAREL** | **USR** |
| Real-time OS |  |  |  |
|  | Create task |  | ✓ |
| File system |  |  |  |
|  | Open/Read/Write file (stdio.h) |  | ✓ |
|  | Check file | ✓ |  |
| WiFi |  |  |  |
|  | Connect |  | ✓ |
|  | Disconnect |  | ✓ |
|  | Get connection status |  | ✓ |
| 2G |  |  |  |
|  | Connect |  | ✓ |
|  | Disconnect |  | ✓ |
|  | Get connection status |  | ✓ |
|  | Receive SMS |  | ✓ |
| NTP client |  |  |  |
|  | Get UTC time |  | ✓ |
| MQTT client |  |  |  |
|  | Connect |  | ✓ |
|  | Disconnect |  | ✓ |
|  | Publish |  | ✓ |
|  | Subscribe |  | ✓ |
|  |  |  |  |
| MQTT payload manager |  |  |  |
|  | Compose payload | ✓ |  |
|  | Parse payload | ✓ |  |
| RS485 driver |  |  |  |
|  | Configure serial |  | ✓ |
|  | Write single/multiple data |  | ✓ |
|  | Read single/multiple data |  | ✓ |
|  |  |  |  |
| Modbus master RTU |  |  |  |
|  | Read/write single/multiple coil |  | ✓ |
|  | Read/write single/multiple HR |  | ✓ |
|  | Read/write single/multiple IR |  | ✓ |
|  | Read/write single/multiple DI |  | ✓ |
|  | Manage report slave id  (raw data) |  | ✓ |
|  | File transfer |  | ✓ |
| Polling engine |  |  |  |
|  | Sample variables (low/high frequency) | ✓ |  |
|  | Sample alarms | ✓ |  |
|  | Start polling | ✓ |  |
|  | Stop polling | ✓ |  |
|  | Reset buffer record | ✓ |  |
|  | Get status slave device | ✓ |  |
| TLS 1.2 library |  |  |  |
|  | Init |  | ✓ |
| https client |  |  |  |
|  | Connect |  | ✓ |
|  | Disconnect |  | ✓ |
|  | Post |  | ✓ |
|  | Get |  | ✓ |
|  | GetResponse |  | ✓ |
| dhcp client |  |  |  |
|  |  |  | ✓ |
| dns client |  |  |  |
|  |  |  | ✓ |
| LED driver |  |  |  |
|  | On |  | ✓ |
|  | Off |  | ✓ |
| Button |  |  |  |
|  | GetStatus |  | ✓ |
| Timer |  |  |  |
|  | TimerStart | ✓ |  |
|  | TimerReset | ✓ |  |
|  | TimerElapsed | ✓ |  |

* 1. Data structures

This part is under traslation

In file data\_types\_IS.h some data types are defined and should be adapted to USR compiler. For ease of convenience, we set the following conventions (that may be modified to adapt to your needs):

/\* you MUST change the data type to match the type your \*/

/\* compiler data types \*/

typedef unsigned char C\_BYTE; // 8 bit value

typedef unsigned char C\_CHAR; // 8 bit value

typedef int C\_INT16; // 16-bit value

typedef unsigned int C\_UINT16; // 16-bit value

typedef long C\_INT32; // 32-bit value

typedef unsigned long C\_UINT32; // 32-bit value

typedef unsigned long C\_WORD; // 32-bit value

/\* floating point IEEE 754 little endian 4 bytes \*/

typedef float C\_FLOAT; //32-bit value

In data\_types\_CAREL.h we defined some useful data types,as below:

#define C\_TIME C\_INT32

typedef C\_BYTE C\_IPV4[4];

typedef C\_BYTE C\_USERNAME[34];

typedef C\_BYTE C\_PASSWORD[32];

typedef C\_BYTE C\_URI[64];

typedef C\_INT16 C\_RES;

typedef C\_CHAR C\_MQTT\_TOPIC[64];

typedef C\_BYTE C\_GUID[16];

* 1. Initialization procedure

### Initialization of operating system

Initialization of OS is completely up to USR. For convenience, we assume it is done calling something like:

void RTOS\_Init();

### Initialization of file system

File system initialization is up to USR and must include all the actions required to properly initialize file system.

This function must be implemented by USR and, once run, allows Carel to access file system using common stdio library functions (such as fopen, fread, fseek, fwrite, fclose, …).

For convenience, we assume it is done calling something like:

File\_System\_Init();

### Initialization of button/leds

Button must be initialized calling a function that, besides configuring GPIO pins, starts a task to monitor button all through the GME life:

Button\_Init();

If the button is pressed when the task is started, a timer is started to check if button is still pressed after 5 seconds. If so, the following function must be called to reset parameters at its factory status:

Set\_Factory();

whose implementation is up to Carel.

Further, during regular operation, button management task must ensure that, if pressed for more than 2s and less than 5s, system is rebooted.

As regards leds, they will be fully managed by USR. USR will be able to obtain required information calling some Carel-provided functions. Carel will implement some functions that return the status of the device to be shown by leds. In detail:

C\_RES Get\_MQTT\_Connection\_Status();

C\_RES Get\_Modbus\_Connection\_Status ();

C\_RES Is\_Model\_Present();

### Recovery of configuration/model information

Configuration and model parameters will be loaded to statically allocated memory structures of type configuration\_file and model\_file (described above) calling:

C\_RES File\_System\_Config\_Load(C\_CHAR\* CFile);

C\_RES File\_System\_Model\_Load(C\_CHAR \* MFile);

Parameters are the names of the configuration and model files respectively. Implementation of these functions is up to Carel.

### Connection to wifi access point/2G provider

In case of WiFi model, GME can simultaneously act as both station (STA) and access point (AP).

In case of STA mode, connection must be established towards an AP, whose ssid and psk can be retrieved calling:

C\_CHAR\* Get\_WiFi\_Station\_SSID();

C\_CHAR\* Get\_WiFi\_Station\_PSK();

which respectively return a pointer to a C\_CHAR buffer of size 30 containing ssid and psk of the AP.

IP address acquisition can be performed in dhcp mode or statically. The choice between the 2 is read with:

C\_BYTE Get\_WiFi\_Station\_DHCP\_mode();

which returns 0 if no such info is available in configuration file, 1 if dhcp mode is to be used and 2 if address will be assigned statically.

In latter case, static address must be read using:

C\_IPV4 Get\_WiFi\_Station\_StaticIP();

which returns a char pointer of size 15 containing the static IP address selected in configuration file. Similarly, netmask and gateways IP can be read from:

C\_IPV4 Get\_WiFi\_Station\_Netmask();

C\_IPV4 Get\_WiFi\_Station\_Gateway();

In case of GME\_AP mode, the GME acts as an AP, having ssid and psk that can be retrieved calling:

C\_CHAR\* Get\_WiFi\_AP\_SSID();

C\_CHAR\* Get\_WiFi\_AP\_PSK();

Its IP and netmask address is read using:

C\_IPV4 Get\_WiFi\_AP\_IP();

C\_IPV4 Get\_WiFi\_AP\_Netmask();

In case of GSM, the GME attempts to connect to 2G network using the APN indicated in configuration file and retrieving such info using function:

C\_CHAR\* Get\_GSM\_APN();

which returns an array of 30 chars containing the APN name.

All the above functions will be implemented by Carel. USR will implement the functions required to connect to networks using the information obtained from the above functions. We assume connection is handled by calls to:

WiFi\_Init();

GSM\_Init();

### Initialization of dhcp client

In case dhcp mode must be active, GME must start a dhcp client, retrieving information from configuration file with the functions described in previous section.

### Initialization of dns client

DNS client must be initialized by USR with dedicated functions.

The IP address of the DNS server to be used can be retrieved from configuration file with:

C\_IPV4 Get\_DNS();

returning a pointer to a char array containing the IP address.

### Initialization of RTC

URL of the NTP server and the port used for NTP connection are written in configuration file. Both can be read from configuration file calling:

C\_URI\* Get\_NTP\_Server();

C\_UINT16 Get\_NTP\_Port();

returning an a string of 64 chars for the server name and a C\_UINT16 containing the port respectively. These functions will be implemented by Carel.

Initialization must be implemented by USR in function:

C\_RES RTC\_Init();

that will recover ntp server name and port and start connection and return SUCCESS as soon as time is available.

### Initialization of certificates

Provided that time is properly retrieved from network, certificate validity can be checked calling:

C\_RES File\_System\_Check\_Cert(C\_CHAR\* file);

that returns SUCCESS if certificate is validated.

### Initialization of TLS library

TLS 1.2 library must be supported and initialized.

### Initialization of https client

A https client must be initialized, with default port 443.

### Initialization of MQTT connection

Connection towards a MQTT server must be initialized.

GME uses parameters in configuration file or, if they are not available, it uses default parameters to finally get actual parameters. Carel will implement functions to retrieve useful information, such as:

C\_URI Get\_MQTT\_broker();

C\_UINT16 Get\_MQTT\_port();

returning the URI of the MQTT broker and the MQTT port, respectively.

Authentication parameters are also taken from configuration file, as:

C\_USERNAME Get\_MQTT\_username();

C\_PASSWORD Get\_MQTT\_password();

### Initialization of RS485

The port used for communication must be selected reading corresponding data in configuration file:

C\_BYTE Get\_RS485\_TTL();

The above returns 0 if Modbus communication must be done on the 3-way connector, and returns 1 if TTL connector must be selected.

Initialization of the serial RS485 port will be implemented by USR calling:

C\_RES RS485\_Init();

This function must recover info on baud rate and data format from model and configuration file, calling:

C\_UINT32 Get\_BaudRate();

C\_BYTE Get\_StopBits();

C\_BYTE Get\_Parity();

Which respectively return baud rate, number of stop bits and parity of the serial data. Number of data bits is set at 8. Supported baud rates must be 4800, 9600, 19200, 38400 and 115200bps, number of stop bits may be 1 or 2 and parity may be even, odd or none.

### Initialization of Modbus

GME acts as Modbus RTU master and can only speak to a single slave at a time.

GME has always the same Modbus address (quale???) and cannot be changed. Modbus address of the connected device is not written in any file but must be recovered from the GME. GME is asked (via MQTT, with a specific message) to perform a scan of the serial line to find the address of the connected device. Scan is done sending a 0x11 command (Report Slave Id), which is managed by Carel products and containing information that help identifying the product itself. When GME detects an answer, it stops scanning the serial line and assumes that all forthcoming communications will be addressed to the just identified address.

If it is required to change the address of the Modbus slave, GME should be rebooted.

### Initialization of Polling engine

GME initializes the polling engine. It gathers the information about the set of Modbus variables that will be monitored (types and indices) from the model file. It allocates the memory buffers required to store Modbus samples in double copy. With double copy it is easier to detect changes in variables that need to be transferred to the MQTT broker. Initialization is up to Carel and will be implemented in:

Polling\_Engine\_Init();

After initialization, polling engine is not yet started: it will be as soon as MQTT connection is established. Considering that polling engine may be occasionally stopped to let other actions occur, a semaphore mechanism must be implemented, using a global variable Stop\_Polling that prevents polling engine from running when set to true.

### Initialization flow

See main\_CAREL.c in the source code folder.

* 1. Regular operation procedure

Periodically, the following operations must be executed:

1. Send status information about connected device via MQTT to the configured broker
2. Process incoming MQTT messages
3. In case polling is enabled, poll variables and, if it’s time to send values via MQTT, do that,
4. If polling is disabled, evaluate other actions: is a file to be transferred? Are there any MQTT-variable changes to be performed?

Based on the above, the rest of the paragraph gives more detail on needed files and functions.

### File system

It may occasionally happen that a new file has to be written in file system. In such occasion, USR can call dedicated functions, depending whether the file is a configuration, model or certificate respectively:

C\_RES File\_System\_Configuration\_Store(C\_CHAR\* name, C\_BYTE\* stream);

C\_RES File\_System\_Model\_Store(C\_CHAR\* name, C\_BYTE\* stream);

C\_RES File\_System\_Cert\_Store(C\_CHAR\* name, C\_BYTE\* stream);

where name is the name of the file to be written and stream is a buffer containing the file.

### Button and leds

During regular operation button and leds has the behaviour described in section dealing with initialization.

### RTC

Time synchronization to the NTP server must be guaranteed at least once a day by a dedicated task implemented by USR.

In any moment, it must be possible to get current UTC time with a call to:

C\_TIME RTC\_Get\_UTC\_Current\_Time(void);

which returns an epoch time (seconds from 1/1/1970), UTC-aligned.

Implementation of this function is up to USR.

Carel will usethe above function to implement timer functionalities.

### Polling engine

Polling engine, that is the mechanism that regulates sampling times and variable set to be polled by GME, is up to Carel. In other words, Carel is responsible for managing the polling engine that periodically gathers data from the connected device.

A variables must be sent to the MQTT broker every time a change occurs. So, the engine must keep a dual copy to easily detect changes and force transmission of changed variables. Variables are sent with a precise temporization.

GME periodically polls variables on the connected device via Modbus, saves read variables to the dedicated buffer and compares it to the previous copy. If a change is detected, varied data is sent to the MQTT broker with well-defined periodicity.

In particular, variable changes will be notified to the MQTT broker with 2 different periodicities: default is that high frequency polling is done every minute, whereas low frequency polling is done every 10 minutes. Alarms are sent as soon as possible. Polling frequencies can be changed with a specific MQTT message.

Polling engine can be started calling

void PollEngine\_\_MBStart(void \*arg);

### Modbus

Modbus communication must be implemented by USR, using proprietary or standard implementations. Carel needs from USR the prototypes of the functions for reading and writing coils, input registers, discrete inputs and holding registers. This way, Carel will implement Modbus polling mechanism independently.

### MQTT

MQTT communication must be implemented by USR, using proprietary or standard implementations. Carel needs from USR the prototypes of the functions for initializing communication, starting/stopping the communication engine, subscribing /unsubscribing to a topic and publishing a topic.

* 1. File transfer on GME

### Certificate update

When a new certificate is to be downloaded, the GME will be notified by the MQTT broker with an update certificate request. The request itself contains the data required for the download (user name, password, uri).

When this request is received, the GME will call

C\_RES HttpsClient\_GetCertificate(C\_USERNAME usr, C\_PASSWORD pwd, C\_URI url);

that will substitute the old certificate with the newly received one.

### Model update

When a new model is available for the GME, the MQTT broker will send a “set device configuration” request. The request contains authentication data and a uri to download the file. When GME receives this request, it has to:

1. Stop polling (if active)
2. Flush away all sampled data
3. Download the file via https calling:

C\_RES HttpsClient\_GetModelFile(C\_USERNAME usr, C\_PASSWORD pwd, C\_URI url);

This function updates all memory structures containing model info

1. When transfer is over, GME must reboot (to let new model become effective).

### FW update for connected device

The GME can upgrade FW of the connected target.

To this aim, Modbus standard command “Write File Record” (0x15) is used.

As above, GME receives a MQTT request with command “update device firmware”, with authentication info for download. When such message is received, GME:

1. Stops Modbus polling engine (if active)
2. Flushes already sampled data
3. Downloads the file via HTTPS and transfers it to connected device in separate chunks. In case of error, GME acts some retries as from standard and, if operation cannot be finished, the whole sequence will be aborted
4. Waits a number of seconds as define in payload request and restarts polling.

In case of wrong file transfer or if FW is not properly installed, connected device will no more answer to Modbus messages and connected device will be marked as offline. A new FW update operation can be started by the MQTT broker.