CAREL – Confidential



**REQUIREMENTS SPECIFICATION**

*Framework for*

*Gateway Middle End*

rev. See history

DRAFT

Index

Revision

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Rev. date | Author | Note |
| *0.1* |  | *V.Cellini* | 1st draft |
| 0.2 |  | A.Bilato | Added some note |
| 0.3 |  | A.Bilato | Added “Data alignment” |
| 0.4 | 15/06/2020 | A.Bilato | Content revision |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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 a 3rd party and Carel will be able to drive them, when needed, by calling some primitives (e.g. MQTT communication primitives will be implemented by 3rd party, 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 3rd party 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 3rd party will just need to be sure that the proper initialization function is called).

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 3rd party. 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 version 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

GTW000M2W0 = the GME-2G+WiFi 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

Important note.  
Out of the box the GME is not able to be immediately operative, these because some initialization task are request cloud side to instruct the GME to operate correctly.

### 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 9 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 10s), system must be rebooted.

If, at power up, it is pressed for more than 5s, GME will boot with the internal bootloader, in this conditions due to the countermeasure that prevent a direct cyber attach the system accept a maximum of 3 upload, this limit is not by passable is intrinsic of the chip.

At running time the button, if pressed for more than 10 sec. e confirmed within 5 sec. when the red led blink slowly, reset the connection parameters.  
This reset do not restore the original FW is an update of that was done in the past.

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

1. is off if not connected to an AP.
2. is always on Green if connected to CAREL MQTT server and configured.
3. is always on Red if connected to CAREL MQTT server and configured.

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. (GPRS) is blink if the connection with the GSM provider was successfully performed, is driven directly from M95.
2. (WORK) blinks when GME receives data via RS485  
   This is useful to detect communication trouble
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).

### Recovery of configuration/model information

Model file can be updated during regular operation via OTA upload.  
The model could be updated at any time via OTA.

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 retrieved from the configuration file, these data are stored encrypted in the GME.

At first boot, AP authentication data are absent, and the system ask for the definition of it.

### 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.

The way to accomplish this configuration may be through a custom part number, and a new configuration file with the APN of the provider choosed by the customer.

### 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 via OTA.  
  
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.

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 at first use, load the default parameters from configuration file, connect to MQTT server and wait for a change credential message.  
The MQTT server accept only the connection of the GME previously registered through the QR code or manually with the data present on the label.

### Initialization of RS485

The serial RS485 port must be initialized at through OTA from the portal, these include 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 which 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.

* 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.

**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.

**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 (\*)serial is provided.   
A demo in source code “C” or Python of a Windows application that perform the serial upgrade must be provided.

(\*) see the above limitation

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** |
| 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 |

Data alignment

Some data structure must be aligned to work in the right way, these structure are easily identifiable due to the fact that a pre compiler directive

#pragma pack(1)

…

#pragma pack()

is placed immediately before/after the structure, for sure could be changed if the compiler in use a different convention.

* 1. Initialization procedure

### Initialization of certificates

If time is properly retrieved from network, certificate validity check must be called.  
If the certificate is not valid the 2nd one will be tried.  
If the GME is unable to use both certificate the only way to restore the device is through a wired update, take care of this because only 3 update must be possible through the serial connection.

### 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 RS485

The port used for communication must be selected reading corresponding data in configuration file, the GME out of the box start with RS485 enabled.

### 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 is specified via OTA  
from the portal.  
Through the portal is possible to change the Modbus, communication parameters, baud rate, bits, and the address of the attached device.

Is also possible for the cloud to ask the GME 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

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.   
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.

If the model file will be changed via OTA the GME require a reboot, this is required because the system need to allocate the new amount of memory in a static way.

### 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.

### 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 is guaranteed at least once a day.  
The time is managed as epoch time (seconds from 1/1/1970), UTC-aligned.

### Polling engine

A variables is 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.  
The high frequency polling support a maximum rate of 1 minute.

* 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).

### 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
4. 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.

### MQTT message and commands

The GME can send receive the data encoded in CBOR through MQTT.  
The message list and the coding of each message are available in <payload_examples> folder.  
The CBOR coding is translated to JSON format on cloud side, this is the reason of the representation you will find for every message, on the left the JSON on the right the CBOR data type. Note that the short description ie. “ver” is maintained so that a CBOR stream is understandable a little also via notepad.

### Model format

The GME can read a model file that meet the specification defined on   
<GME_Modbus_scan_datatable_Flash_ipotesi_2.xlsx>

The maximum number of variables are 200 or alternatively the number that is possible to store in 2KB of file size. The reason is the system is flexible and accept different combinations of variables, but is limited to 2KB of flash to store the file and 200 variables limits due to the RAM availability.