CAREL – Confidential



**REQUIREMENTS SPECIFICATION**

*Framework for*

*Gateway Middle End*

rev. 0.1

DRAFT

Index

[Index 2](#_Toc19535512)

[Revision 3](#_Toc19535513)

[1 Introduction 4](#_Toc19535514)

[1.1 Scope of the document 4](#_Toc19535515)

[1.2 Definitioni, acronimi e abbreviazioni 4](#_Toc19535516)

[2 Operating principle 5](#_Toc19535517)

[2.1 Initialization 5](#_Toc19535518)

[2.2 Regular operation 6](#_Toc19535519)

[2.3 File transfer on GME 7](#_Toc19535520)

[3 Implementation 8](#_Toc19535521)

[3.1 Real-time OS 9](#_Toc19535522)

[3.2 File system 9](#_Toc19535523)

Revision

|  |  |  |  |
| --- | --- | --- | --- |
| Rev. | Rev. date | Author | Note |
| *0.1* |  | *V.Cellini* | 1st draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

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
* temporarily keeps data and sends it to a MQTT broker (2G/WiFi)
* gets requests of changes of variable values from the MQTT broker and resends them to the connected device
* gets a file name and url (containing model info) via MQTT and accepts the file itself via HTTPS (for FW update).

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. To this extent, Carel will provide documentation describing the functions to be called to implement different functions. In some cases, these functions will not be fully implemented but they will require HW/platform specific additional implementations to be filled in by USR.

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.

TODO

* 1. Definitioni, acronimi e abbreviazioni

AP = WiFi access point

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\_TABLE = tabella che definisce il significato dei vari registri Modbus

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. During file system initialization, all the actions required to initialize file system must be carried on, so a bunch of primitive must be available to store/recover information.

### 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.

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 and password for connection
* MQTT broker, port, user and password

Default configuration has:

* baud rate 19200bps

Configuration file cannot be updated.????

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 has:

* data format 8-n-1

Model file can be updated during regular operation.

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/2G provider

GME must connect to a wifi access point or to a 2G provider, depending on the HW on which we’re running.

In case of WiFi, TODO

In case of GSM, TODO

### 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.  
  
AB: Alternatively for the 2G model is possible to use the time retrieved from the GSM network and adapted to return a UTC value. This in the case you have serious memory constraint and the space for the NTP library is not compatible with the available memory.

VC: Yes, in mobile connections there are other ways to get time (<https://www.raviyp.com/embedded/244-gsm-network-time-synchronization-for-rtc-using-at-clts-command>) BUT what do we do in case of WiFi?

### Certificate check

The GME must natively contain 2 certificate files. 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.  
AB: The GME usually use the 1st certificate if this certificate is invalid or refused from the server, the GME try to use the 2nd certificate to do the same.  
This method give to CAREL the possibility to substitute one certificate via OTA.

### 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).

AB: The baud rate coming from the cloud the data bits/stop bits/parity are stored inside the model file, so that without a model file the GME use the default values 19200,8,1,N.  
(vedi sopra in verde)

### Initialization of Modbus slave communication

All buffers required to store Modbus samples before sending must be statically allocated, polling engine is not yet started: it will be as soon as MQTT connection is established.

### 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.

Button is intended as a 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, TODO

### Initialization of https/dhcp/dns client

An https client must be initialized.

A dhcp client must be initialized.

A DNS client must be initialized. Two DNS names are available in configuration file.

TODO

### 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.

* 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 only 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 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.

AB: About the possibility to perform a GME FW updates a good choice if possible is to use

HTTPS file transfer to upgrade the GME itself, in this way all the files to the GME are transferred in the same way, is it possible ?

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 RTS\_\*.\* 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 |  | ✓ |
|  | 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 | ✓ |  |
|  |  |  |  |
| Polling engine |  |  |  |
|  | Sample variables (low/high frequency) | ✓ |  |
|  | Sample alarms | ✓ |  |
|  | Start polling | ✓ |  |
|  | Stop polling | ✓ |  |
|  | Reset buffer record | ✓ |  |
|  | Get status slave device | ✓ |  |
| https client |  |  |  |
|  | Connect |  | ✓ |
|  | Disconnect |  | ✓ |
|  | Post |  | ✓ |
|  | Get |  | ✓ |
|  | GetResponse |  | ✓ |
| dhcp client |  |  |  |
|  |  |  | ✓ |
| dns client |  |  |  |
|  |  |  | ✓ |
| LED driver |  |  |  |
|  | On |  | ✓ |
|  | Off |  | ✓ |
| Button |  |  |  |
|  | GetStatus |  | ✓ |
| Timer |  |  |  |
|  | TimerStart | ✓ |  |
|  | TimerReset | ✓ |  |
|  | TimerElapsed | ✓ |  |

* 1. Data structures

The following data structures are required to store useful data:

mqtt\_cfg

TODO

* 1. Init procedure

/\* RTOS initialization \*/

RTOS\_Init();

/\* File system initialization \*/

File\_System\_Init();

/\* Check and load configuration \*/

if(File\_System\_Config\_Check(“config.bin”) == SUCCESS)

File\_System\_Config\_Load(“config.bin”);

else

File\_System\_DefaultConfig\_Load();

/\* Connect to 2G provider or WiFi access point according to part number \*/

#if (NETWORK\_INTERFACE == WIFI\_INTERFACE)

WiFi\_Init(“your\_ap”,”psk”);

#elif (NETWORK\_INTERFACE == GSM\_INTERFACE)

GSM\_Init(“your.apn.com”);

#endif

/\* Clock initialization and certificate validation \*/

if(RTC\_Init(ntp\_server, port) == SUCCESS)

if(File\_System\_Check\_Cert(“cert1.crt”) == SUCCESS)

File\_System\_Cert\_Load(“cert1.crt”);

else if(File\_System\_Cert\_Check(“cert2.crt”) == SUCCESS)

File\_System\_Cert\_Load(“cert2.crt”);

else

???? //no valid certificate

else

???? // time cannot be retrieved, what to do?

/\* Check and load model \*/

if(File\_System\_Check\_File(“model.bin”) == SUCCESS)

File\_System\_Model\_Load(“model.bin”);

else

File\_System\_DefaultModel\_Load();

/\* RS485 initialization \*/

RS485\_Init(baud, nbit, parity, stop);

/\* Polling engine initialization, temporarily stopped \*/

Stop\_Polling = TRUE;

Polling\_Engine\_Init();

/\* GPIO initialization\*/

LED\_Init();

Button\_Init();

/\* MQTT connection initialization and polling start \*/

if(MQTT\_Init(&mqtt\_cfg) == SUCCESS)

STOP\_Polling=FALSE;

* 1. Regular operation procedure

Periodically, the following operations must be executed:

MQTT\_Send\_Status();

MQTT\_Incoming\_message();

if (STOP\_Polling == FALSE){

Polling\_Engine();

if(Timer\_Elapsed(MQTT\_SEND\_TIME))

MQTT\_Send\_Message();

}

else{

file transfer?

Variable change from mqtt?

}

*USB??? (vedi document che descrive l’aggiornamento ota)*

*Non mi è chiaro cosa succede se il file di modello non è valido*

*E se non è valido quello di configurazione?*

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

* 1. Real-time OS
  2. File system

Stdio library functions fopen, fread, fseek, fwrite, fclose must be supported.

In File\_System\_IS.h, a file system initialization must be managed.

Carel will implement all the functions needed to validate configuration and model files, and to load these settings (or default ones) to statically allocated memory structures.

Certificate validation?

* 1. MQTT Initialization