EVERLASTING

Electric Vehicle Enhanced Range, Lifetime And Safety Through INGenious battery management

D6.5 – BMS Application Programming Interface

August 2019

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 713771

Confidential

PROJECT SHEET	
Project Acronym	**EVERLASTING**
Project Full Title	Electric Vehicle Enhanced Range, Lifetime And Safety Through INGenious battery management
Grant Agreement	**713771**
Call Identifier	H2020-GV8-2015
Topic	GV-8-2015: Electric vehicles' enhanced performance and integration into the transport system and the grid
Type of Action	Research and Innovation action
Project Duration	48 months (01/09/2016 – 31/08/2020)
Coordinator	VLAAMSE INSTELLING VOOR TECHNOLOGISCH ONDERZOEK NV (BE) - *VITO*
Consortium Partners	COMMISSARIAT A L ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES (FR) - *CEA* SIEMENS INDUSTRY SOFTWARE SAS (FR) - *Siemens PLM* TECHNISCHE UNIVERSITAET MUENCHEN (DE) - *TUM* TUV SUD BATTERY TESTING GMBH (DE) - *TUV SUD* ALGOLION LTD (IL) - *ALGOLION LTD* RHEINISCH-WESTFAELISCHE TECHNISCHE HOCHSCHULE AACHEN (DE) - *RWTH AACHEN* LION SMART GMBH (DE) - *LION SMART* TECHNISCHE UNIVERSITEIT EINDHOVEN (NL) - *TU/E* VOLTIA AS (SK) - *VOLTIA* VDL ENABLING TRANSPORT SOLUTIONS (NL) – *VDL ETS*
Website	www.everlasting-project.eu

DELIVERABLE SHEET	
Title	**D6.5 – BMS Application Programming Interface**
Related WP	WP6 (Standardized architecture)
Lead Beneficiary	LION SMART GMBH
Author(s)	Prashanth Vemireddy (LION SMART GMBH)
Reviewer(s)	Everhard Ros (LION SMART GMBH) Sebastian Ludwig (TUM)
Type	Report
Dissemination level	CONFIDENTIAL (only for members of consortium – incl. commission services)
Due Date	August 31, 2019
Submission date	August 31, 2019
Status and Version	Final, version 1.0

REVISION HISTORY			
Version	**Date**	**Author/Reviewer**	**Notes**
V0.1	29/08/2019	Prashanth Vemireddy (LION Smart)	Final draft
V0.2	30/08/2019	Sebastian Ludwig (TUM)	Peer review
V0.3	30/08/2019	Everhard Ros (LION Smart)	Internal review
V1.0	31/08/2019	Carlo Mol (VITO) Coordinator	Submission to the EC

DISCLAIMER

The opinion stated in this report reflects the opinion of the authors and not the opinion of the European Commission.

The commercial use of any information contained in this document may require a license from the owner of that information.

All EVERLASTING consortium members are committed to publish accurate information and take the greatest care to do so. However, the EVERLASTING consortium members cannot accept liability for any inaccuracies or omissions nor do they accept liability for any direct, indirect, special, consequential or other losses or damages of any kind arising out of the use of this information.

ACKNOWLEDGEMENT

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 713771

EXECUTIVE SUMMARY

As part of its WP6 activities in the Horizon 2020-Project EVERLASTING, the partner LION Smart will develop a standardized Battery Management System (BMS) hardware and software architecture within an automotive context.

The purpose of this document is to consolidate the requirements of the involved stakeholders, together with the currently considered architectural variants and technical solutions, in order to further plan and concretize the development of this BMS.

This reflects both internal analysis in LION Smart GmbH and with the involved project partners, ALGOLION, RWTH Aachen, TUM and VOLTIA.

The plan is to approve (without or with requested modifications) this joint collection and second version with the project partners. Once the formalized set of requirements at system level has been approved by the involved partners, LION Smart will proceed with the definitive design and construction of the BMS final demonstrator.

TABLE OF CONTENTS

List of Abbreviations and Acronyms

Acronym	Definition
API	Application Programming Interface
BMS	Battery Management System
DOW	Description of Work
LCM	LION Control Module
LMM	LION Measurement Module
LMM-E	LION Measurement Module for EVERLASTING
tbd	To be decided
WP	Work package
WPL	Work package leader

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The goal of this document is to disclose the software Application Programming Interface (API) access available for the BMS proposed in the document D6.7 'BMS standardization proposal'. The need of a BMS which can accommodate applications to be run (within permissible resource utility) architectural design of the demonstrator BMS. It is based on the BMS standard architecture proposal. This document details the design specifications, which are consecutively integrated into the demonstrator.

This includes in particular the definition of APIs for the interaction and the integrated operation of the different layers of the Battery Management System. For the Low-Level Management Layer, sets of APIs will be developed for the implementation of safety features, data transmission and SOC/SOH estimation algorithms. Particular attention will be given to the design of the data transmission APIs as they constitute the entities for the transmission of safety configuration settings and functions within this level. Transmitting cell measurement data and run-time settings for the estimation algorithms to external databases or cloud services are going to be additional features implemented in the data transmission APIs.

The tasks that lead to this document are introduced in the following chapter. One of the objectives of this project is to reduce the cost of operation through software algorithms. This deliverable is addressing an essential component of integrating partner software algorithms on BMS. This document would provide comprehension of public access software application peripheral interface to run other third-party applications.

1 BMS SOFTWARE

1.1 SOFTWARE ARCHITECTURE

The following chapter introduces software architecture of the demonstrator BMS. Figure 2 represents the organization of the BMS functions. Figure 1 represents the software architecture layered block diagram. The following chapter focus on the block, 'API for Applications'.

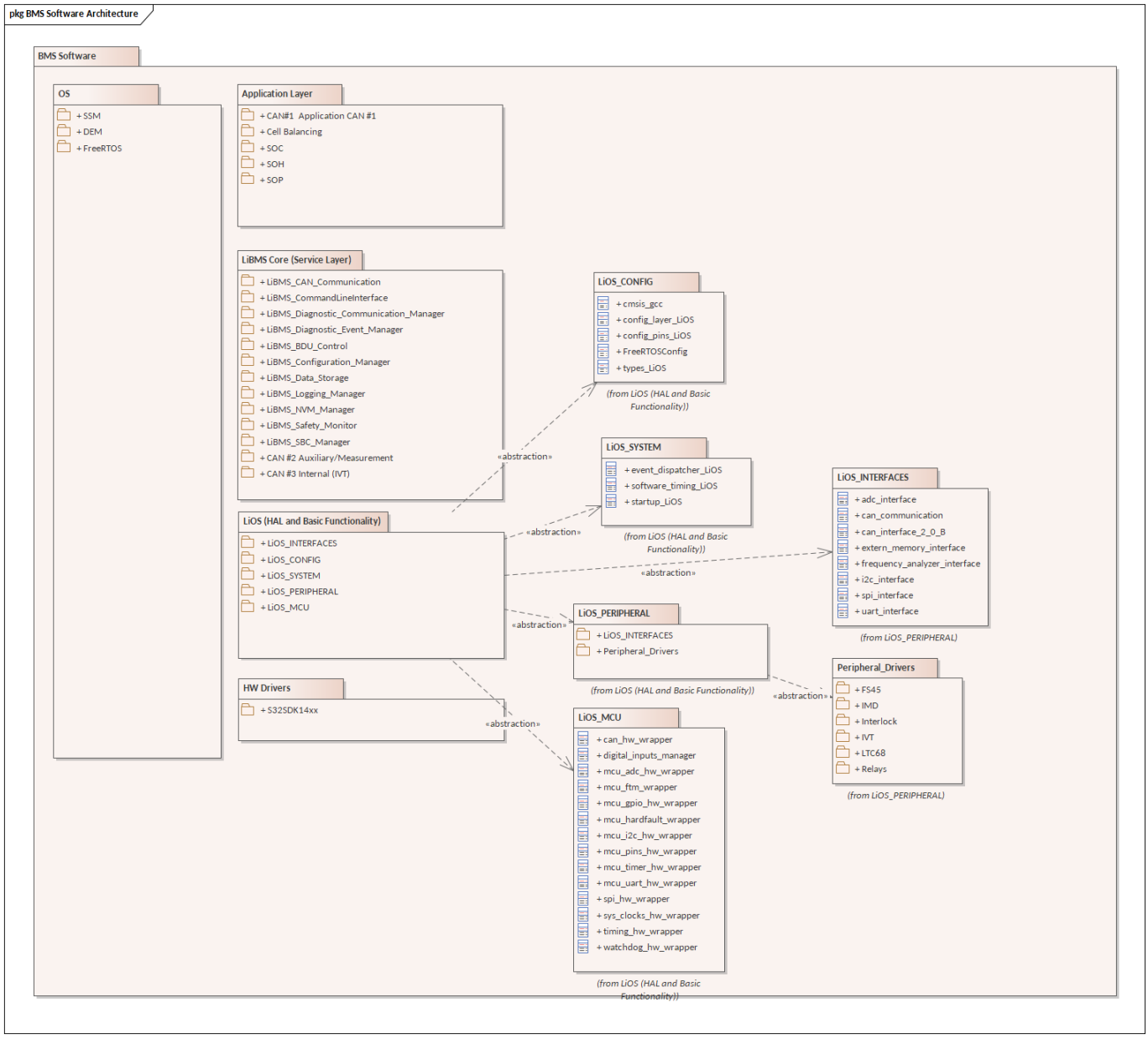

Figure 1: Block diagram of Software architecture

The various components illustrated in the block diagram depicted in Figure 1 are listed and described in the following subsections. An overview of architecture and the dissemination of components is shown below.

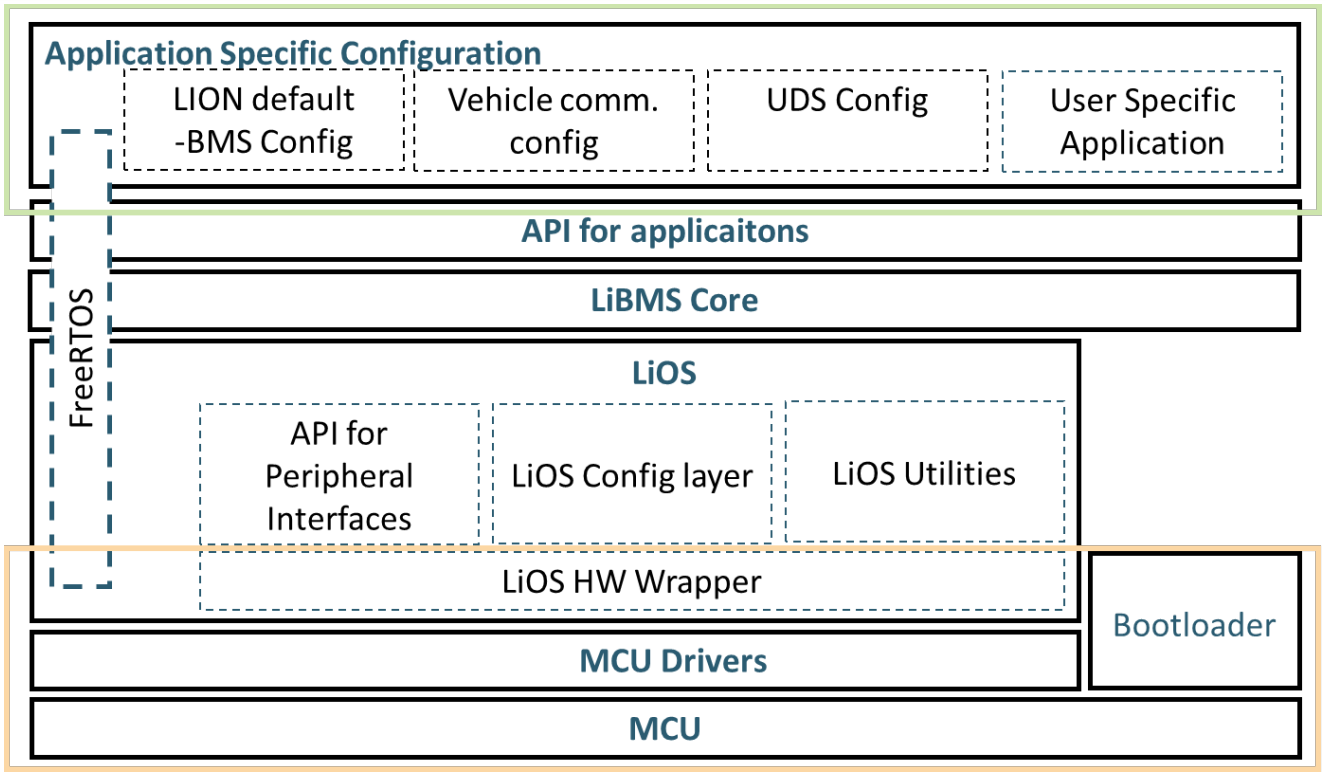

Figure 2: Software Architecture layer diagram

1.1.1 OS

These are the packages that provide the core OS features such as scheduling tasks, mutexes, queues and the main System State Machine (SSM). The Diagnostic Event Manager (DEM) is responsible for managing various high-level system events.

1.1.2 APPLICATION LAYER

Application layer contains all the higher-level applications. The third-party applications should be integrated at this level. More detailed description of integration into this layer can be found in chapters 3 and 4.

1.1.3 LIBMS CORE

The packages located in this layer provide higher-level BMS functionalities. They interact with the software packages on the hardware layer to control the hardware interfaces indirectly.

1.1.4 LIOS

The LiOS layer incorporates software packages which directly interact with the SBC and the other hardware devices on the LCM and LMM, such as the ADC and SPI among others.

1.2 Architectural Principles

1.2.1 General

Software architecture is developed in focus to provide the following features,

1. Modularity
2. Flexibility,
3. Reusability and
4. Interfaces to run partner applications

The software development cycle is followed with V-Model approach for being able to vividly track software units and their tests, and software integration and tests.

BMS peripherals are measurement and control systems. Some of them are listed as below,

1) HV current and voltage sensors,
2) high-side drivers to control contactors and
3) isolation monitoring device etc.

The BMS software does provide the functionality listed below,

a) Primary functionality,
   i) monitor cell voltages and temperatures,
   ii) control its peripherals,
   iii) control contactors, and
   iv) perform safety calculations and decisions
b) Perform statistical calculations
   i) cell Voltage statistics
   ii) cell temperatures statistics
   iii) cell balancing strategy,
   iv) SOX (SOC, SOP, SOH etc.)
c) External communication and storage interfaces
   i) CAN
   ii) Non-volatile memory
d) Data storage interfaces
   i) cells information
   ii) BMS peripherals information

Communication interfaces (external): The primary functions of the BMS can be observed from an external debug CAN communication channel or accessing an on-board non-volatile memory unit.

1.2.2 API ACCESS DESIGN

This subchapter represents the framework provided to bridge the gap from BMS to other applications to be run on BMS.

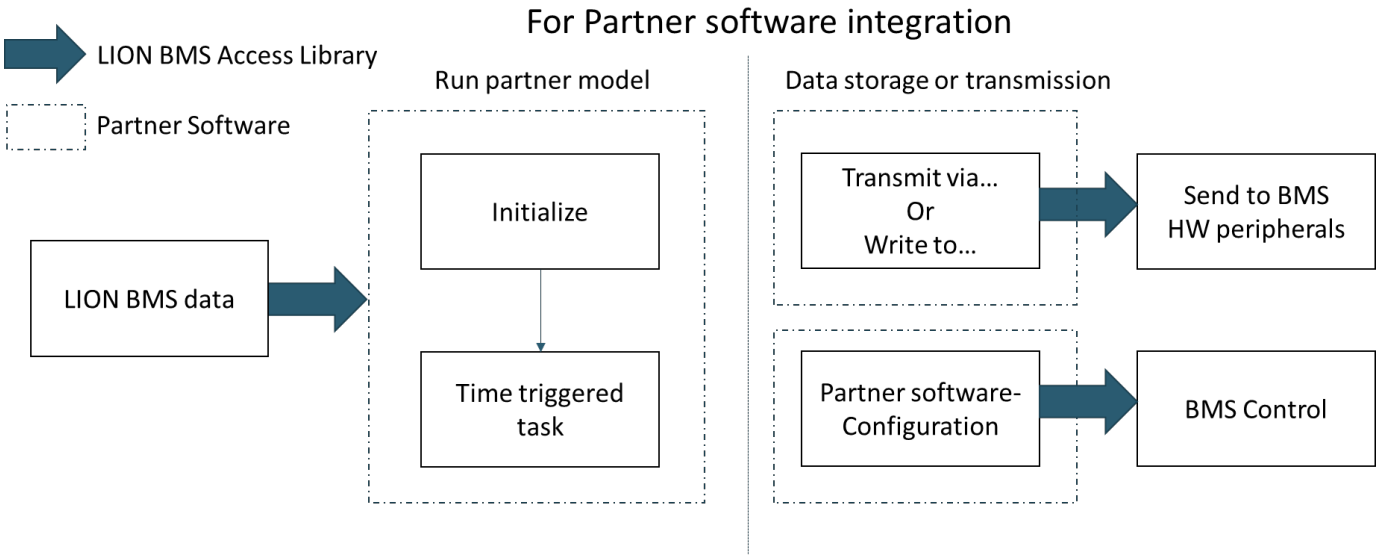

Figure 3: Integral and bridging parts of software API

Additionally, terminate function would also be treated. Data input and peripheral access are described in Chapter 3 and Chapter 4. The set of functions accessible by authorized applications is shown in table below.

Functions	
APPL_InitializationFunction()	Initialization function of the application
APPL_PeriodicOneStepFunction()	Step function which would be called periodically
APPL_EventTriggeredFunction()	Event based function, called upon a criterion
APPL_TerminateFunction()	Function to stop the application

Table 1: Important functions for running applications

2 Specific Requirements from Partners

The section reflects discussions with the involved project partners, ALGOLION, RWTH Aachen and TUM.

In addition to the requirements jointly developed by the involved partners, a survey with open questions was circulated and answered as it can be found in annex 1.

2.1 Requirements from TUM

1) Models
   a) For Reduced Order Models
      i) Initialization
         (1) Cell voltage
         (2) Cell temperature
      ii) Periodic input data
         (1) Current
         (2) Absolute time in millisecond
      iii) Local data storage for next iteration or output:
         (1) State Matrix
         (2) Solid phase parameters
   b) For Load management
      i) Periodic input
         (1) Cell voltage
         (2) Balancing resistance
         (3) Balancing time
      ii) Output
         (1) Integrated balancing load
2) Peripheral access interface requirements
      i) CAN interface to send data out from BMS

2.2 Requirements from ALGOLION

1) Models
   i) Battery health status
   ii) Initialization
   iii) Initialize Health Status
2) Periodic/time-driven execution
   i) Get Current Health Status
3) Input data
   i) All cell voltages
   ii) All cell temperatures
   iii) Battery pack current
   iv) Coulomb counting
b) Power
c) Cell balancing information
4) Peripheral access interface requirements
   i) CAN interface to send data out
   ii) Non-volatile memory access

3 Master Application Programming Interface

This chapter details APIs for integrating applications into LCM software.

3.1 Data structures

```
struct CanMessage_FD
{
    uint32_t    cs;             /*!< Code and Status*/
    uint32_t    msgId;          /*!< Message Buffer ID*/
    uint8_t     data[64];       /*!< Bytes of the FlexCAN message*/
    uint8_t     dataLen;        /*!< Length of data in bytes */
    uint8_t     rtr;            /*!< Remote Request Bit */
    uint8_t     controllerNo;   /*!< CAN controller no. the message was received or
                                     should be transmitted starting with 1 */
}__PACKED;
```

3.2 Enumerations

enum BMS_Control_States

```
SSM_STATE_POWER_UP              = 1,
SSM_STATE_SYSTEM_INITILIZING    = 2,
SSM_STATE_HV_OFF_NORMAL         = 3,
SSM_STATE_HV_PRECHARGE          = 4,
SSM_STATE_HV_ON                 = 5,
SSM_STATE_HV_OFF_URGENT_PRE     = 6,
SSM_STATE_HV_OFF_URGENT         = 7,
SSM_STATE_SYSTEM_SHUTDOWN       = 8,
SSM_STATE_START_BALANCING       = 9,
SSM_SLEEP_FINAL                 = 10,
SSM_STATE_FAILSTATE             = 11
```

3.3 Cell and Battery Pack Data Access Functions

uint16_t LMM_GetCalculatedMinCellVoltagePerModule (uint8_t moduleNumber)

"LMM_GetCalculatedMinCellVoltagePerModule" provides the minimum of the measured cell voltages in the module.

Parameters:

in	moduleNumber	module number of the queried module

Returns:

returns value of minimum of the measured cell voltages of the queried module.

uint16_t LMM_GetCalculatedMaxCellVoltagePerModule (uint8_t moduleNumber)

"LMM_GetCalculatedMaxCellVoltagePerModule" provides the maximum of the measured cell voltages in the module.

Parameters:

in	moduleNumber	module number of the queried module

Returns:
returns value of maximum of the measured cell voltages of the queried module.

uint16_t LMM_GetCalculatedMeanCellVoltagePerModule (uint8_t moduleNumber)

"LMM_GetCalculatedMeanCellVoltagePerModule" provides the mean of the measured cell voltages in the module.

Parameters:

in	moduleNumber	module number of the queried module

Returns:
returns value of mean of the measured cell voltages of the queried module.

uint16_t LMM_GetCalculatedMinCellTemperaturPerModule (uint8_t *moduleNumber*)

"LMM_GetCalculatedMinCellTemperaturPerModule" provides the minimum of the measured cell temperatures in the module.

Parameters:

in	*moduleNumber*	The module number of the queried module

Returns:
returns value of minimum of the measured cell temperatures of the queried module.

uint16_t LMM_GetCalculatedMaxCellTemperaturPerModule (uint8_t moduleNumber)

"LMM_GetCalculatedMaxCellTemperaturPerModule" provides the maximum of the measured cell temperatures in the module.

Parameters:

in	moduleNumber	module number of the queried module

Returns:
returns value of maximum of the measured cell temperatures of the queried module.

uint16_t LMM_GetCalculatedMeanCellTemperaturPerModule (uint8_t *moduleNumber*)

"LMM_GetCalculatedMeanCellTemperaturPerModule" provides the mean of the measured cell temperatures in the module.

Parameters:

in	*moduleNumber*	The module number of the queried module

Returns:
returns value of mean of the measured cell temperatures of the queried module.

uint16_t LMM_GetCalculatedMinPCBTemperaturPerModule (uint8_t moduleNumber)

"LMM_GetCalculatedMinPCBTemperaturPerModule" provides the minimum of the measured temperatures of PCB in module.

Parameters:

in	moduleNumber	module number of the queried module

Returns:
returns value of minimum of the measured LMM PCB temperatures of the queried module.

uint16_t LMM_GetCalculatedMaxPCBTemperaturPerModule (uint8_t moduleNumber)

"LMM_GetCalculatedMaxPCBTemperaturPerModule" provides the maximum of the measured temperatures of PCB in module.

Parameters:

in	moduleNumber	module number of the queried module

Returns:

returns value of maximum of the measured LMM PCB temperatures of the queried module.

uint16_t LMM_GetCalculatedMeanPCBTemperaturPerModule (uint8_t moduleNumber)

"LMM_GetCalculatedMeanPCBTemperaturPerModule" provides the mean of the measured temperatures of PCB in module.

Parameters:

in	moduleNumber	module number of the queried module

Returns:

returns value of maximum of the measured LMM PCB temperatures of the queried module.

const uint16_t* LMM_GetCellVoltagesPerModule (uint8_t *moduleNumber*)

"LMM_GetCellVoltagesPerModule" provides a pointer to access cell voltages of the module

Parameters:

in	*moduleNumber*	The module number of the queried module

Returns:

returns 16-bit pointer address of the first cell voltage in the module.

const uint16_t* LMM_GetCellTemperaturesPerModule (uint8_t *moduleNumber*)

"LMM_GetCellTemperaturesPerModule" provides a pointer to access cell temperatures of a module

Parameters:

in	*moduleNumber*	The module number of the queried module

Returns:

returns 16-bit pointer address of the first cell temperature in the module.

uint16_t LMM_GetMinCellVoltagePerPack (void)

"LMM_GetMinCellVoltagePerPack" provides the minimum of the measured cell voltages in the battery pack.

Returns:

returns value of mean of the measured cell voltages of the battery pack.

uint16_t LMM_GetMaxCellVoltagePerPack (void)

"LMM_GetMaxCellVoltagePerPack" provides the maximum of the measured cell voltages in the battery pack.

Returns:
returns value of maximum of the measured cell voltages of the battery pack.

uint16_t LMM_GetMeanCellVoltagePerPack (void)

"LMM_GetMeanCellVoltagePerPack" provides the mean of the measured cell voltages in the battery pack.

Returns:
returns value of mean of the measured cell voltages of the battery pack.

uint16_t LMM_GetMinCellVoltageIndexPerPack (void)

"LMM_GetMinCellVoltageIndexPerPack" returns index of the cell with minimum voltage of the cells in the whole battery pack

Returns:
Returns index value of the cell with minimum voltage.

uint16_t LMM_GetMaxCellVoltageIndexPerPack (void)

"LMM_GetMaxCellVoltageIndexPerPack" returns index of the cell with maximum voltage of the cells in the whole battery pack

Returns:
Returns index value of the cell with maximum voltage.

uint16_t LMM_GetMinCellTemperaturePerPack (void)

"LMM_GetMinCellTemperaturePerPack" provides the minimum of the measured temperatures of battery pack.

Returns:
returns minimum temperature of the cells in the battery pack.

uint16_t LMM_GetMaxCellTemperaturePerPack (void)

"LMM_GetMaxCellTemperaturePerPack" provides the maximum of the measured temperatures of battery pack.

Returns:
returns maximum temperature of the cells in the battery pack.

uint16_t LMM_GetMeanCellTemperaturePerPack (void)

"LMM_GetMeanCellTemperaturePerPack" provides the mean of the measured temperatures of battery pack.

Returns:
returns mean temperature of the cells in the battery pack.

uint16_t LMM_GetMinCellTemperatureIndexPerPack (void)

"LMM_GetMinCellTemperatureIndexPerPack" returns index of the cell with minimum temperature of the cells in the whole battery pack.

Returns:
Returns index value of the cell with minimum temperature.

uint16_t LMM_GetMaxCellTemperatureIndexPerPack (void)

"LMM_GetMaxCellTemperatureIndexPerPack" returns index of the cell with maximum temperature of the cells in the whole battery pack.

Returns:
Returns index value of the cell with maximum temperature.

uint32_t LMM_GetBalancingStatus (uint8_t moduleNumber)

" LMM_GetBalancingStatus " returns current cell balancing configuration of the module.

Parameters:

in	*moduleNumber*	The module number of the queried module

Returns:
Returns 32-bit value 32 bits per each module. Bit 0 to bit 31 corresponds to cell 0 to cell 31.

uint32_t LMM_GetSumCellVoltagePerPack (void)

"LMM_GetSumCellVoltage" returns sum of voltages of the cells in the battery pack

Returns:
Returns sum of cell voltages of the battery pack.

uint32_t BP_GetBatteryPackSoC (void)

"BP_GetBatteryPackSoC" provides real battery pack SoC.

Returns:

returns uint16 real battery pack SoC.

uint32_t BP_GetUsableSoC (void)

"BP_GetUsableSoC" provides usable battery pack SoC.

Returns:

returns uint16 usable battery pack SoC.

enum BMS_Control_States SSTM_GetCurrentStmState(void)

"BP_GetUsableSoC" provides current BMS state from main statemachine.

Returns:

returns enum BMS_Control_States.

3.4 Hardware Peripheral Service Access Functions

bool CAN_CU_TransmitMessage (struct CanMessage_FD* pCanMessage)

" **CAN_CU_TransmitMessage** " transmit data through CAN hardware peripheral.

Parameters:

in	pCanMessage	Can message payload with header and tail.

Returns:

returns bool successful or not successful.

bool NVM_ WriteData(uint32_t dataSegment, uint8_t* ptrData, dataLength)

" **NVM_ StorageData**" write data into non-volatile memory.

Parameters:

In	dataSegment	Data segment code.
in	ptrData	Pointer to the data to be written in non-volatile memory storage.
in	dataLength	Length of data to be written in bytes.

Returns:

returns bool successful or not successful.

bool NVM_ ReadData(uint32_t dataSegment, uint8_t* ptrData, dataLength)

" **NVM_ StorageData**" read data from non-volatile memory.

Parameters:

In	dataSegment	Data segment code.
out	ptrData	Pointer to the data to which date from non-volatile memory storage is written into.
in	dataLength	Length of data to be written in bytes.

Returns:

returns bool successful or not successful.

4 Slave Application Programming Interface

This chapter details APIs for integrating applications into LMM-E software.

4.1 Data structures

```
struct CanMessage
{
    uint32_t    id;           /*!< Message Identifier */
    uint8_t     data[8];      /*!< Data buffer*/
    uint8_t     length;       /*!< Data length (0-8) */
    uint8_t     rtr;          /*!< Remote Request Bit */
    uint8_t     controllerNo; /*!< CAN controller no. the message was received or should be
                                   transmitted starting with 1 */
}__PACKED;
```

4.2 Cell Data Access Functions

const uint16_t* LMM_GetCellVoltages(void)

"LMM_GetCellVoltages" provides a pointer to access cell voltages of the module

Returns:

returns 16 bit pointer address of the first cell voltage in the module.

const uint16_t* LMM_GetCellTemperatures (void)

"LMM_GetCellTemperatures" provides a pointer to access cell temperatures of a module

Returns:

returns 16 bit pointer address of the first cell temperature in the module.

uint32_t LMM_GetBalancingStatus (uint8_t moduleNumber)

" LMM_GetBalancingStatus " returns current cell balancing configuration of the module.

Parameters:

in	*moduleNumber*	The module number of the queried module

Returns:

Returns 32-bit value 32 bits per each module. Bit 0 to bit 31 corresponds to cell 0 to cell 31.

uint32_t BP_GetBatteryPackSoC (void)

"BP_GetBatteryPackSoC" provides real battery pack SoC.

Returns:

returns uint16 real battery pack SoC.

uint32_t BP_GetUsableSoC (void)

"BP_GetUsableSoC" provides usable battery pack SoC.

Returns:

returns uint16 usable battery pack SoC.

enum BMS_Control_States SSTM_GetCurrentStmState(void)

"BP_GetUsableSoC" provides current BMS state from main statemachine.

Returns:

returns enum BMS_Control_States.

4.3 Hardware Peripheral Service Access Functions

bool CAN_TransmitMessage (struct CanMessage* pCanMessage)

" **CAN_TransmitMessage** " transmit data through CAN hardware peripheral.

Parameters:

in	pCanMessage	Can message payload with header and tail.

Returns:

returns bool successful or not successful.

bool NVM_ WriteData(uint32_t dataSegment, uint8_t* ptrData, dataLength)

" **NVM_ StorageData**" write data into non-volatile memory.

Parameters:

In	dataSegment	Data segment code.
in	ptrData	Pointer to the data to be written in non-volatile memory storage.
in	dataLength	Length of data to be written in bytes.

Returns:

returns bool successful or not successful.

bool NVM_ ReadData(uint32_t dataSegment, uint8_t* ptrData, dataLength)

" **NVM_ StorageData**" read data from non-volatile memory.

Parameters:

In	dataSegment	Data segment code.
out	ptrData	Pointer to the data to which date from non-volatile memory storage is written into.
in	dataLength	Length of data to be written in bytes.

Returns:

returns bool successful or not successful.

CONCLUSIONS

In the scope of development of BMS for demonstrator, a BMS had been successfully developed. The important aspects of software API are presented. Data and peripheral access functions are described. This public document is further developed exhaustively to be use by partners in the scope EVERLASTING project.

REFERENCES

[1] J. M. Alvarez, *D6.3 – Closed source hardware design and protoype.*
[2] P. Vemireddy and B. Jayaraman, *D6.7 BMS standardization Proposal,* 2019.