Multi-Modal Intelligent Traffic Signal System (MMITSS) Phase III:

Task 3.1 Specification for J2735 Message Library

Version 1.1

November 14, 2018

University of Arizona (Lead)

University of California PATH Program

RECORD OF CHANGES

A – Added, M- Modified, D - Deleted

Version	Date	Description
1.0	10/12/2018	Initial
1.1	11/14/2018	Incorporated comments from MMITSS team

Table of Contents

Table of Figures

List of Tables

1 Introduction

In the development of MMITSS development plan, a review of the current MMITSS prototypes has resulted in identification of several common libraries/components that can be abstracted from the current code base and then updated/improved in the modified code base and used by both the AZ and CA prototypes. Figure 1 illustrates these common components that will support both the MMITSS-AZ and the MMITSS-CA prototypes, as well as other Connected Vehicle (CV) applications that could be developed in the same framework.

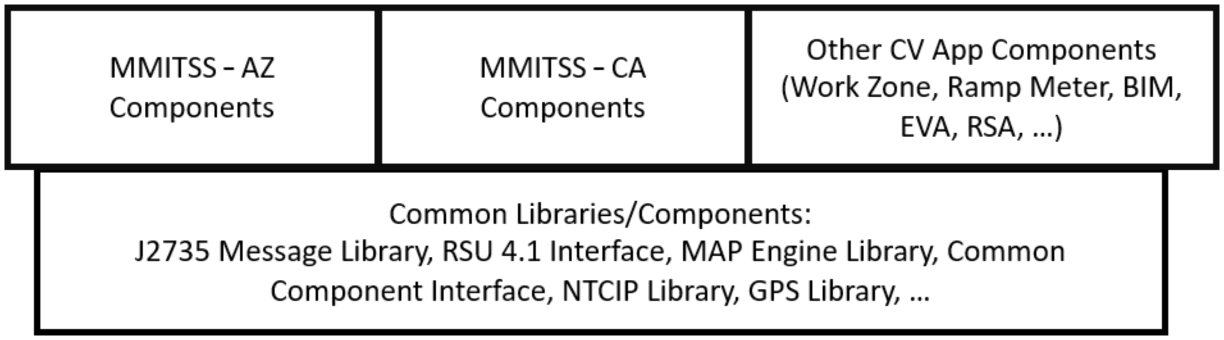
MMITSS - AZ Components
MMITSS - CA Components
Other CV App Components (Work Zone, Ramp Meter, BIM, EVA, RSA, ...)
Common Libraries/Components: J2735 Message Library, RSU 4.1 Interface, MAP Engine Library, Common Component Interface, NTCIP Library, GPS Library, ...

Figure 1: Common MMITSS Libraries/Components

1.1 Overview of J2735 Message Library

The common J2735 Message Library should be able to compile and link by other MMITSS software modules on both the stand-alone Linux-like MRP (MMITSS Roadside Processor) computer and vendor specific On-Board Unit (OBU) to encode and decode MMITSS relevant SAE J2735-201603 messages, including BSM, SRM, MAP, SPaT, SSM, and relevant RTCM Corrections messages.

Figure 2 is a basic Context Diagram of J2735 Message Library (JML). The JML provides a Message Encoder API which takes MMITSS internal data object as the input and outputs the UPER encoding of the SAE J2735-2016 message payload for transmitting over-the-air, and a Message Decoder API which takes the SAE J2735-2016 message payload as the input and assign the decoded results to the MMITSS internal data object.

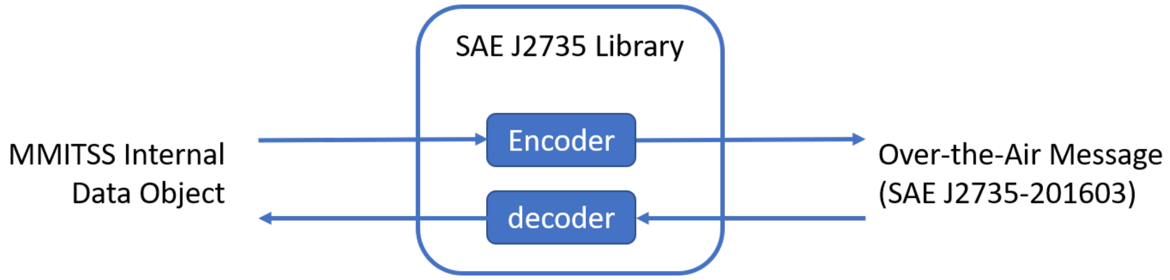
SAE J2735 Library
Encoder
decoder
MMITSS Internal Data Object
Over-the-Air Message (SAE J2735-201603)

Figure 2: Context Diagram of J2735 Message Library

1.2 Open Source ASN.1 C Compiler

SAE J2735 messages are defined in ASN.1 format[1]. The open source ASN.1 C compiler (i.e., *asn1c*)[2] is utilized for converting the ASN.1 message definition into a set of *.[ch]* files and for generating UPER encoding and decoding. The JML includes the *.[ch]* files and UPER encoding and decoding functions of *asn1c* and the additional translation between *asn1c* data objects and MMITSS internal data objects.

1.3 MMITSS Required SAE J2735 Data Elements

ASN.1 definition for MMITSS relevant SAE J2735 messages includes Required and Optional Data Frames (DFs) and Data Elements (DEs). Some of the optional Data Frames and/or Data Elements are required to support MMITSS applications. The MMITSS team has identified MMITSS required SAE J2735 data elements. See Appendix A for more information.

1.4 MMITSS Internal Data Objects for J2735 Message Library

To reduce the reliance and the needs for re-development of MMITSS due to the changes of SAE J2735 messages in the future, the MMITSS team has defined the internal data structures used as inputs for message encoding and outputs for message decoding, therefore, the software modules that consume the over-the-air messages can be independent of SAE J2735 message definition. The internal data objects for MMITSS relevant SAE J2735 messages are included in Appendix B. Since the encoding and decoding of MAP payload are included in the MAP Engine Library, the internal data objects defined in Appendix B does not include MAP object.

1.5 Default Value of SAE J2735 Required Data Elements

Every data element of a BSM Part I (i.e., BSMcoreData) is required in transmission. However, not every data element is available on the OBU. Some data elements, such as transmission state and steering wheel angle, require the access to the vehicle's CAN Bus. SAE J2735 standard defines special values to indicate that these data elements are unknow/unavailable. The SAE J2945/1 standard[3] defines the Minimum Transmission Criteria (MINTX) for BSM Part I, which specifies which data elements can be set to unavailable/unknown value. This requires the JML to provide *reset*, *set*, and *get* functions to set the value of possible unavailable data element to the default "unavailable" value (*reset* function) or set the value to sensor data (*set* function) before UPER encoding, and to get the value of possible unavailable data element (*get* function) after UPER decoding. The *get* function shall return an indication whether the data element is "unavailable" or true sensor data, without knowing the specific unavailable/unknown values per SAE J2735. MMITSS internal data objects presented in Appendix B are defined in such as a way that includes the *reset*, *set*, and *get* functions.

[1] SAE, Dedicated Short Range Communications (DSRC) Message Set Dictionary Set J2735-201603. Available at https://www.sae.org/standards/content/j2735set_201603/.
[2] The ASN.1 Compiler – asn1c. Available at https://github.com/vlm/asn1c.
[3] SAE J2945/1, On-Board System Requirements for V2V Safety Communications. March 2016.

1.6 Basic Functions

The JML is identical for MRP and for OBU. Figure 3 illustrates the basic functions and activities of JML highlighting Inputs on MRP and OBU.

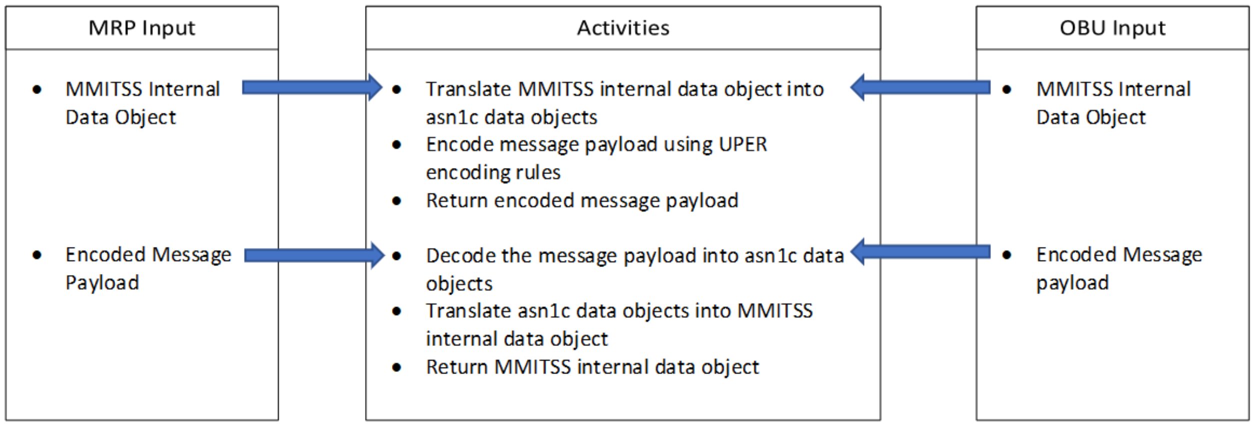
MRP Input
• MMITSS Internal Data Object
• Encoded Message Payload
Activities
• Translate MMITSS internal data object into asn1c data objects
• Encode message payload using UPER encoding rules
• Return encoded message payload
• Decode the message payload into asn1c data objects
• Translate asn1c data objects into MMITSS internal data object
• Return MMITSS internal data object
OBU Input
• MMITSS Internal Data Object
• Encoded Message payload

Figure 3: Basic Functions and Activities of J2735 Message Library

1.7 Operating System

The same JML should be able to create and decode properly formatted UPER messages on both the MRP and OBU. The MRP is a stand-alone Linux-like computer that utilizes an embedded Linux operating system. Each OBU is vendor specific and has an OBU SDK (Software Development Kit) which is required to compile the JML. Different OBU vendors have different versions of the SDK. Since both AZ and CA utilize Savari OBUs, the development of JML focuses on Ubuntu (MRP) and Savari OBU. The JML implementation might be able to work with OBUs from other vendors, however, the development of JML to work with other vendors' OBU is out of scope of this task. Table 1 lists the requirement on operating system.

Table 1: Requirement on Operating System

Device	Operating System	Version	Note
MRP Computer	Ubuntu	16.04 or later	As of September 2018, the latest Ubuntu is version 18.04.
Savari OBU	Embedded Linux	5.10 or later	

2 Requirements

This section contains the requirements for J2735 Message Library (JML). Requirements listed in this section use the following terminology:

- SHALL: Indicate that the definition is an absolute requirement of the specification.
- SHOULD: Indicates the definition is non-mandatory but recommended.

Table 2: List of Requirements for J2735 Message Library

Req. ID	Category	Description	Apply To	Verification Method
JML-Req-001	Operating System	The JML SHALL be able to be built on Ubuntu 16.04 or later.	MRP	Test
JML -Req-002	Operating System	The JML SHALL be able to be built on Savari OBU with firmware 5.10 or later.	OBU	Test
JML-Req-003	Interface	When populating a MMITSS internal data object, the JML SHOULD provide a *reset* function to automatically assign a possible unavailable data element to its unknown/unavailable value. See comments in *MMITSScommon.h* included in Appendix B for unavailable data elements.	MRP and OBU	Test
JML-Req-004	Interface	When populating a MMITSS internal data object, the JML SHOULD provide a *set* function to assign a possible unavailable data element to the sensor data input. See comments in *MMITSScommon.h* included in Appendix B for unavailable data elements.	MRP and OBU	Test
JML-Req-005	Functional	When provided with a MMITSS internal data object, the JML SHALL perform unit translation per SAE J2735-201603.	MRP and OBU	Test
JML-Req-006	Functional	The JML SHALL properly encode the translated data object into SAE J2735-2016 Message Frame, and return the encoded message. Note: Utilizing USDOT's Message Validator tool[4] to verify the correctness of UPER encoding.	MRP and OBU	Test
JML-Req-007	Functional	When provided with an encoded message payload, the JML SHALL decode the message into a data object.	MRP and OBU	Test
JML-Req-008	Functional	The JML SHALL perform unit translation on the decoded data object, populate and return the corresponding MMITSS internal data object.	MRP and OBU	Test
JML-Req-009	Interface	When accessing a MMITSS internal data object, the JML SHOULD provide a *get* function to access the value of a possible unavailable data element with a returned indication of whether the data element is unavailable/unknow or true sensor data. See comments in *MMITSScommon.h* included in Appendix B for unavailable data elements.	MRP and OBU	Test

[4] USDOT Connected Vehicle Message Validator. https://webapp2.connectedvcs.com/validator/

Appendix A MMITSS Required SAE J2735 Data Elements

This appendix lists MMITSS required SAE J2735-201603 data elements for MMITSS relevant messages.

A.1 Basic Safety Message (BSM)

MSG_BasicSafetyMessage (BSM)		SAE J2735-201603	MMITSS Implementation	Purpose to Include
coreData	BSMcoreData (DF)	Required	Required	
partII	PartIIcontent (DF)	Optional	Optional	

A.2 MAP

MSG_SPAT (SPaT)		SAE J2735-201603	MMITSS Implementation	Purpose to Include
timeStamp	MinuteOfTheYear (DE)	Optional	Optional	
name	DescriptiveName (DE)	Optional	Optional	
intersections	IntersectionStateList (DF)	Required	Required	
IntersectionState (DF)				
name	DescriptiveName (DE)	Optional	Optional	
id	IntersectionReferenceID (DF)	Required	Required	
region	RoadRegulatorID (DE)	Optional	Required	For compatibility across regions
id	IntersectionID (DE)	Required	Required	
revision	MsgCount (DE)	Required	Required	
status	IntersectionStatusObject (DE)	Required	Required	
moy	MinuteOfTheYear (DE)	Optional	Optional	
timeStamp	DSecond (DE)	Optional	Optional	
enabledLanes	EnabledLaneList (DF)	Optional	Optional	
states	MovementList (DF)	Required	Required	
movementName	DescriptiveName (DE)	Optional	Optional	
signalGroup	SignalGroupID (DE)	Required	Required	
state-time-speed	MovementEventList (DF)	Required	Required	
MovementEvent (DF)				
eventState	MovementPhaseState (DE)	Required	Required	
timing	TimeChangeDetails (DF)	Optional	Required	For RLVW and generating SRM
startTime	TimeMark (DE)	Optional	Optional	
minEndTime	TimeMark (DE)	Required	Required	
maxEndTime	TimeMark (DE)	Optional	Optional	
likelyTime	TimeMark (DE)	Optional	Optional	Better to have but require prediction work
confidence	TimeIntervalConfidence (DE)	Optional	Optional	
nextTime	TimeMark (DE)	Optional	Optional	
speeds	AdvisorySpeedList (DF)	Optional	Optional	
maneuverAssistList	ManeuverAssistList (DF)	Optional	Optional	

A.3 Signal Phase and Timing (SPaT)

MSG_SPAT (SPaT)		SAE J2735-201603	MMITSS Implementation	Purpose to Include
timeStamp	MinuteOfTheYear (DE)	Optional	Optional	
name	DescriptiveName (DE)	Optional	Optional	
intersections	IntersectionStateList (DF)	Required	Required	
IntersectionState (DF)				
name	DescriptiveName (DE)	Optional	Optional	
id	IntersectionReferenceID (DF)	Required	Required	
region	RoadRegulatorID (DE)	Optional	Required	For compatibility across regions
id	IntersectionID (DE)	Required	Required	
revision	MsgCount (DE)	Required	Required	
status	IntersectionStatusObject (DE)	Required	Required	
moy	MinuteOfTheYear (DE)	Optional	Optional	
timeStamp	DSecond (DE)	Optional	Optional	
enabledLanes	EnabledLaneList (DF)	Optional	Optional	
states	MovementList (DF)	Required	Required	
movementName	DescriptiveName (DE)	Optional	Optional	
signalGroup	SignalGroupID (DE)	Required	Required	
state-time-speed	MovementEventList (DF)	Required	Required	
MovementEvent (DF)				
eventState	MovementPhaseState (DE)	Required	Required	
timing	TimeChangeDetails (DF)	Optional	Required	For RLVW and generating SRM
startTime	TimeMark (DE)	Optional	Optional	
minEndTime	TimeMark (DE)	Required	Required	
maxEndTime	TimeMark (DE)	Optional	Optional	
likelyTime	TimeMark (DE)	Optional	Optional	Better to have but require prediction work
confidence	TimeIntervalConfidence (DE)	Optional	Optional	
nextTime	TimeMark (DE)	Optional	Optional	
speeds	AdvisorySpeedList (DF)	Optional	Optional	
maneuverAssistList	ManeuverAssistList (DF)	Optional	Optional	

A.4 Signal Request Message (SRM)

MSG_SignalRequestMessage (SRM)		SAE J2735-201603	MMITSS Implementation	Purpose to Include
timeStamp	MinuteOfTheYear (DE)	Optional	Optional	
second	DSecond (DE)	Required	Required	
sequenceNumber	MsgCount (DE)	Optional	Required	For identifying updated SRM
requests	SignalRequestList (DF)	Optional	Required	For priority control
SignalRequestPackage (DF)				
request	SignalRequest (DF)	Required	Required	
id	IntersectionReferenceID (DF)	Required	Required	
region	RoadRegulatorID (DE)	Optional	Required	For compatibility across regions
id	IntersectionID (DE)	Required	Required	
requestID	RequestID (DE)	Required	Required	
requestType	PriorityRequestType (DE)	Required	Required	
inBoundLane	IntersectionAccessPoint (DF)	Required	Required	Choice of LaneID or ApproachID
outBoundLane	IntersectionAccessPoint (DF)	Optional	Optional	Better to have (for priority control)
minute	MinuteOfTheYear (DE)	Optional	Optional	The MRP could estimate the service time and duration by combining SRM with BSM
second	DSecond (DE)	Optional	Optional	
duration	DSecond (DE)	Optional	Optional	
requestor	RequestorDescription (DF)	Required	Required	
id	VehicleID (DE)	Required	Required	BSM TemporaryID
type	RequestorType (DF)	Optional	Required	For multi-modal priority control
role	BasicVehicleRole (DE)	Required	Required	
subrole	RequestSubRole (DE)	Optional	Optional	
request	RequestImportanceLevel (DE)	Optional	Required	For N-level priority control
iso3883	Iso3833VehicleType (DE)	Optional	Optional	
hpmsType	VehicleType (DE)	Optional	Required	For multi-modal priority control
position	RequestorPositionVector (DF)	Optional	Required	Part of BSM
position	Position3D (DF)	Required	Required	
heading	Angle (DE)	Optional	Required	
speed	TransmissionAndSpeed (DF)	Optional	Required	
name	DescriptiveName (DE)	Optional	Optional	
routeName	DescriptiveName (DE)	Optional	Optional	It is OBU's responsibility to determine the level of priority to request.
transitStatus	TransitVehicleStatus (DE)	Optional	Optional	
transitOccupancy	TransitVehicleOccupancy (DE)	Optional	Optional	
transitSchedule	DeltaTime (DE)	Optional	Optional	

A.5 Signal Status Message

MSG_SignalStatusMessage (SSM)		SAE J2735-201603	MMITSS Implementation	Purpose to Include
timeStamp	MinuteOfTheYear (DE)	Optional	Optional	
second	DSecond (DE)	Required	Required	
sequenceNumber	MsgCount (DE)	Optional	Optional	
status	SignalStatusList (DF)	Required	Required	
SignalStatus (DF)				
sequenceNumber	MsgCount (DE)	Required	Required	
id	IntersectionReferenceID (DF)	Required	Required	
region	RoadRegulatorID (DE)	Optional	Required	For compatibility across regions
id	IntersectionID (DE)	Required	Required	
sigStatus	SignalStatusPackageList (DF)	Required	Required	
SignalStatusPackage (DF)				
requester	SignalRequesterInfo (DF)	Optional	Required	For acknowledgement of priority control
id	VehicleID (DE)	Required	Required	
request	RequestID (DE)	Required	Required	
sequenceNumber	MsgCount (DE)	Required	Required	
role	BasicVehicleRole (DF)	Optional	Optional	
typeData	RequestorType (DF)	Optional	Optional	
inboundOn	IntersectionAccessPoint (DF)	Required	Required	These are SRM data elements. SSM to be consistent with SRM.
outboundOn	IntersectionAccessPoint (DF)	Optional	Optional	
minute	MinuteOfTheYear (DE)	Optional	Optional	
second	DSecond (DE)	Optional	Optional	
duration	DSecond (DE)	Optional	Required	
status	PrioritizationResponseStatus (DE)	Required	Required	

A.6 RTCM Correction Messages

MSG_RTCMcorrections (RTCM)		SAE J2735-201603	MMITSS Implementation	Purpose to Include
msgCnt	MsgCount (DE)	Required	Required	
rev	RTCM-Revision (DE)	Required	Required	
timeStamp	MinuteOfTheYear (DE)	Optional	Optional	
anchorPoint	FullPositionVector (DF)	Optional	Optional	
rtcmHeader	RTCMheader (DF)	Optional	Optional	
msgs	RTCMmessageList (DF)	Required	Required	
message	RTCMmessage (DE)	Required	Required	

Appendix B MMITSS Internal Data Objects for J2735 Message Library

```
//*************************************************************************************************************
//
// © 2016-2017 Regents of the University of California on behalf of the University of California at Berkeley
//       with rights granted for USDOT OSADP distribution with the ECL-2.0 open source license.
//
//*************************************************************************************************************
#ifndef _MMITSS_COMMON_H
#define _MMITSS_COMMON_H

#include <cstdint>
#include <bitset>
#include <string>
#include <vector>

namespace MMITSScommon
{
  enum class engageStatus  : uint8_t {unavailable, off, on, engaged};
  enum class transGear     : uint8_t {unavailable, neutral, park, forward, reverse};
  enum class requestType   : uint8_t {unknown, priorityRequest, requestUpdate, priorityCancellation};
  enum class vehicleRole   : uint8_t {unknown, car, transit, emergency, truck, motorcycle, cyclist, pedestrian, roadWork,
    roadRescue};
  enum class vehicleType   : uint8_t {unknown, car, bus, special, truck, other};
  enum class requestStatus : uint8_t {unavailable, requested, processing, watchOtherTraffic, granted, rejected, maxPresence,
    reserviceLocked};

  /// wheelStr2num: covert wheel ("leftFront", "leftRear", "rightFront", or "rightRear") to bit number
  auto wheelStr2num = [](const std::string& wheel)->uint32_t
  {
    uint32_t wheel_num = 0;
    if (wheel.compare("leftFront") == 0)
      wheel_num = 1;
    else if (wheel.compare("leftRear") == 0)
      wheel_num = 2;
    else if (wheel.compare("rightFront") == 0)
      wheel_num = 3;
    else if (wheel.compare("rightRear") == 0)
      wheel_num = 4;
    return(wheel_num);
  };

  /// setValue: set the value of maybe unavailable variable var2set to input value
  template<class T>
  static inline void setValue(bool& isSet, T& var2set, const T& value)
  {
    isSet = true;
    var2set = value;
  };

  /// getValue: get value of maybe unavailable variable var2get
  /// return False if var2get is not available; otherwise return True and var2get value
  template<class T>
  static inline bool getValue(const bool& isSet, const T& var2get, T& value)
  {
    if (isSet)
    {
      value = var2get;
      return(true);
    }
    return(false);
  };

  struct geoPos_t
  {
    double latitude;        // in degrees
    double longitude;       // in degrees
    double elevation;       // in meters
  };

  struct posAccy_t
  {
    double semiMajor;       // in meters
    double semiMinor;       // in meters
    double orientation;     // in degrees
  };

  struct vehAcc_t
  {
    double longitudinal;    // in m/s^2,
    double lateral;         // in m/s^2, may be set to unavailable
    double vertical;        // in m/s^2, may be set to unavailable
    double yawRate;         // in degrees per second
    bool isLatSet;          // True if lateral acceleration is available
    bool isVertSet;         // True if vertical acceleration is available
    void reset(void)
    { /// set lateral and vertical acceleration to unavailable
      isLatSet = false;
      isVertSet = false;
    };
    void setLatAcc(const double& latAcc)
      {MMITSScommon::setValue<double>(isLatSet, lateral, latAcc);};
  void setVertAcc(const double& vertAcc)
    {MMITSScommon::setValue<double>(isVertSet, vertical, vertAcc);};
  bool getLatAcc(double& latAcc)
    {return(MMITSScommon::getValue<double>(isLatSet, lateral, latAcc));};
  bool getVertAcc(double& vertAcc)
    {return(MMITSScommon::getValue<double>(isVertSet, vertical, vertAcc));};
};

struct vehSize_t
{
  double width;           // in meters
  double length;          // in meters
};

struct brakeStatus_t
{
  std::bitset<5> wheelBrakeStatus;
    /*  with bit defined as
     *    unavailable (0) -- When set, the brake applied status is unavailable
     *    leftFront   (1) -- Left Front Active
     *    leftRear    (2) -- Left Rear Active
     *    rightFront  (3) -- Right Front Active
     *    rightRear   (4) -- Right Rear Active
     */
  MMITSScommon::engageStatus tractionControlStatus;
  MMITSScommon::engageStatus absStatus;
  MMITSScommon::engageStatus stabilityControlStatus;
  MMITSScommon::engageStatus brakeBoostApplied;
  MMITSScommon::engageStatus auxiliaryBrakeStatus;
  void reset(void)
  { /// set all brake system status to unavailable
    wheelBrakeStatus.reset();
    wheelBrakeStatus.set(0);
    tractionControlStatus = MMITSScommon::engageStatus::unavailable;
    absStatus = MMITSScommon::engageStatus::unavailable;
    stabilityControlStatus = MMITSScommon::engageStatus::unavailable;
    brakeBoostApplied = MMITSScommon::engageStatus::unavailable;
    auxiliaryBrakeStatus = MMITSScommon::engageStatus::unavailable;
  };
  /* ---------------------------------------------------------------------------------------------
   * setWheelBrakeStatus: set wheelBrakeStatus for individual wheel
   * Input:
   *    wheel  - STRING, choice of "leftFront", "leftRear", "rightFront", or "rightRear".
   *    status - BOOL, True for On, False for Off.
   * Return: BOOL
   *    False if wheel is not "leftFront", "leftRear", "rightFront", or "rightRear";
   *    True otherwise.
   * ---------------------------------------------------------------------------------------------*/
  bool setWheelBrakeStatus(const std::string& wheel, const bool& status)
  { /// wheel: leftFront, leftRear, rightFront, or rightRear
    /// status: True for On, False for Off
    uint32_t wheel_num = MMITSScommon::wheelStr2num(wheel);
    if (wheel_num == 0)
      return(false);
    if (wheelBrakeStatus.test(0))
      wheelBrakeStatus.reset(0);
    if (status)
      wheelBrakeStatus.set(wheel_num);
    else
      wheelBrakeStatus.reset(wheel_num);
    return(true);
  };
  /* ---------------------------------------------------------------------------------------------
   * getWheelBrakeStatus: get wheelBrakeStatus for individual wheel
   * Input: wheel - STRING, choice of "leftFront", "leftRear", "rightFront", or "rightRear".
   * Return: BOOL
   *    False brakeStatus unavailable or if wheel is not one of the choices;
   *    True otherwise and status is set to True/False if brake is/not applied to wheel.
   * ---------------------------------------------------------------------------------------------*/
  bool getWheelBrakeStatus(const std::string& wheel, bool& status)
  {
    if (wheelBrakeStatus.test(0))
      return(false);
    uint32_t wheel_num = MMITSScommon::wheelStr2num(wheel);
    if (wheel_num == 0)
      return(false);
    status = wheelBrakeStatus.test(wheel_num);
    return(true);
  };
};

struct bsm_t
{ /// BSMcoreData
  uint32_t msgCnt;                      // 0..127
  uint32_t id;                          // TemporaryID
  uint32_t timeStampSec;                // milliseconds of the minute
  double speed;                         // in meters per second (mps)
  double heading;                       // in degrees
  double steeringAngle;                 // in degrees, may be set to unavailable
  bool isAngleSet;                      // True if steeringAngle is available
  MMITSScommon::transGear transState;   // may be set to unavailable
  MMITSScommon::geoPos_t  geoPos;
  MMITSScommon::posAccy_t posAccy;
  MMITSScommon::vehAcc_t  vehAcc;       // lateral and vertical acceleration may be set to unavailable
  MMITSScommon::vehSize_t vehSize;
  MMITSScommon::brakeStatus_t brakeStatus; // may be set to unavailable
  /// reset function to set maybe unavailable variables to unavailable
  void reset(void)
  {
    isAngleSet = false;
    transState = MMITSScommon::transGear::unavailable;
    vehAcc.reset();
    brakeStatus.reset();
  };
  /// set functions to assign value to maybe unavailable variables
  void setSteeringAngle (const double& angle)
    {MMITSScommon::setValue<double>(isAngleSet, steeringAngle, angle);};
  void setLatAcc(const double& latAcc)
    {vehAcc.setLatAcc(latAcc);};
  void setVertAcc(const double& vertAcc)
    {vehAcc.setVertAcc(vertAcc);};
  bool setWheelBrakeStatus(const std::string& wheel, const bool& status)
    {return(brakeStatus.setWheelBrakeStatus(wheel, status));};
  /// get functions to get value of maybe unavailable variables. Return False if the variable is not available.
  bool getSteeringAngle(double& angle)
    {return(MMITSScommon::getValue<double>(isAngleSet, steeringAngle, angle));};
  bool getLatAcc(double& latAcc)
    {return(vehAcc.getLatAcc(latAcc));};
  bool getVertAcc(double& vertAcc)
    {return(vehAcc.getVertAcc(vertAcc));};
  bool getWheelBrakeStatus(const std::string& wheel, bool& status)
    {return(brakeStatus.getWheelBrakeStatus(wheel, status));};
};

struct serviceTime_t
{
  uint32_t ETAminute;       // minutes of the year
  uint32_t ETAsec;          // milliseconds of the minute
  uint32_t duration;        // in milliseconds
};

struct srm_t
{
  uint32_t msgCnt;                     // 0..127
  uint32_t timeStampMinute;            // (OPTIONAL) minutes of the year
  uint32_t timeStampSec;               // milliseconds of the minute
  bool isMinuteOfYearSet;              // True if timeStampMinute is set
  /// intersection data to request
  MMITSScommon::requestType reqType;
  uint32_t regionalId;
  uint32_t intId;
  uint32_t reqId;
  uint32_t reqLevel;                   // 0 = unknown
  uint32_t inApprochId;                // 0 = unknown
  uint32_t inLaneId;                   // 0 = unknown
  uint32_t outApproachId;              // (OPTIONAL) 0 = unknown
  uint32_t outLaneId;                  // (OPTIONAL) 0 = unknown
  MMITSScommon::serviceTime_t serviceTime; // (OPTIONAL)
  bool isServiceTimeSet;               // True if serviceTime is set
  /// vehicle data (see bsm_t)
  uint32_t vehId;
  double   speed;                      // in meters per second (mps)
  double   heading;                    // in degrees
  MMITSScommon::transGear transState;  // may be set to unavailable
  MMITSScommon::geoPos_t  geoPos;
  MMITSScommon::vehicleRole vehRole;
  MMITSScommon::vehicleType vehType;
  uint32_t num_axle;                   // for truck, 2..7
  /// reset function to set maybe unavailable or OPTIONAL variables to unavailable/unknown
  void reset(void)
  {
    isMinuteOfYearSet = false;
    isServiceTimeSet = false;
    reqType = MMITSScommon::requestType::unknown;
    transState = MMITSScommon::transGear::unavailable;
    vehRole = MMITSScommon::vehicleRole::unknown;
    vehType = MMITSScommon::vehicleType::unknown;
    num_axle = 2;
  };
  /// set functions to assign value to OPTIONAL variables
  void setMinuteOfYear(const uint32_t& ts)
    {MMITSScommon::setValue<uint32_t>(isMinuteOfYearSet, timeStampMinute, ts);};
  void setServiceTime(const MMITSScommon::serviceTime_t& reqTime)
    {MMITSScommon::setValue<MMITSScommon::serviceTime_t>(isServiceTimeSet, serviceTime, reqTime);};
  /// get functions to get value of OPTIONAL variables. Return False if the variable is not available.
  bool getMinuteOfYear(uint32_t& ts)
    {return(MMITSScommon::getValue<uint32_t>(isMinuteOfYearSet, timeStampMinute, ts));};
  bool getServiceTime(MMITSScommon::serviceTime_t& reqTime)
    {return(MMITSScommon::getValue<MMITSScommon::serviceTime_t>(isServiceTimeSet, serviceTime, reqTime));};
};

struct requetStatus_t
{
  uint32_t  vehId;
  uint32_t  reqId;
  uint32_t  sequenceNumber;    // msgCnt in SRM
  uint32_t  inApprochId;       // 0 = unknown
  uint32_t  inLaneId;          // 0 = unknown
  uint32_t  outApproachId;     // 0 = unknown
  uint32_t  outLaneId;         // 0 = unknown
  MMITSScommon::serviceTime_t serviceTime; // (OPTIONAL)
  bool isServiceTimeSet;       // True if serviceTime is set
  MMITSScommon::vehicleRole vehRole;
  MMITSScommon::requestStatus status;
  void reset(void)
  {
    isServiceTimeSet = 0;
    vehRole = MMITSScommon::vehicleRole::unknown;
    status  = MMITSScommon::requestStatus::unavailable;
  };
  /// set functions to assign value to OPTIONAL variables
  void setServiceTime(const MMITSScommon::serviceTime_t& reqTime)
    {MMITSScommon::setValue<MMITSScommon::serviceTime_t>(isServiceTimeSet, serviceTime, reqTime);};
  /// get functions to get value of OPTIONAL variables. Return False if the variable is not available.
  bool getServiceTime(MMITSScommon::serviceTime_t& reqTime)
    {return(MMITSScommon::getValue<MMITSScommon::serviceTime_t>(isServiceTimeSet, serviceTime, reqTime));};
};
```

```
struct ssm_t
{
  uint32_t  timeStampMinute;   // (OPTIOANL) minutes of the year
  uint32_t  timeStampSec;      //  milliseconds of the minute
  uint32_t  msgCnt;            // (OPTIONAL) 0..127
  uint32_t  updateCnt;         // (0..127), change whenever mpRequetStatus has changed
  uint16_t  regionalId;
  uint16_t  id;                // intersection ID
  bool isMinuteOfYearSet;      // True if timeStampMinute is set
  bool isMsgCountSet;          // True if msgCnt is set
  std::vector<MMITSScommon::requetStatus_t> mpRequetStatus;  // maximum entries: 32
  /// reset function to set OPTIONAL variables to unavailable/unknown
  void reset(void)
  {
    isMinuteOfYearSet = false;
    isMsgCountSet = false;
    mpRequetStatus.clear();
  };
  /// set functions to assign value to OPTIONAL variables
  void setMinuteOfYear(const uint32_t& ts)
    {MMITSScommon::setValue<uint32_t>(isMinuteOfYearSet, timeStampMinute, ts);};
  void setMsgCount(const uint32_t& cnt)
    {MMITSScommon::setValue<uint32_t>(isMsgCountSet, msgCnt, cnt);};
  /// get functions to get value of OPTIONAL variables. Return False if the variable is not available.
  bool getMinuteOfYear(uint32_t& ts)
    {return(MMITSScommon::getValue<uint32_t>(isMinuteOfYearSet, timeStampMinute, ts));};
  bool getMsgCount(uint32_t& cnt)
    {return(MMITSScommon::getValue<uint32_t>(isMsgCountSet, msgCnt, cnt));};
};
```