OBU-201U API Guide

HW Version: 0B
U-Boot: 1.3.3
SDK: 4.11.0-sc
Stack: US, 914

V1.0

Regulatory Information

Export Control Notice

Unex strives to make each customer's experience as smooth as possible while complying with global trade compliance laws and regulations. In pursuit of these goals and to ensure that you understand that by purchasing V2X communication solution (including but not limit to VTX-201, PCB-201, OBU-201, OBU-201E, OBU-201U, HVP-201, HRU-201, HRU-201E, HRU-201U) as well as follow-up models from Unex, they may become subject to export/import control laws of the U.S. or other countries, we hereby notify you of the following:

The commodities, technical data, or software of V2X communication solution (including but not limit to VTX-201, PCB-201, OBU-201, OBU-201E, OBU-201U, HVP-201, HRU-201, HRU-201E, HRU-201U) as well as follow-up models that you have purchased or will purchase from Unex Technology Corporation or its subsidiaries or affiliates ("Unex"), as well as other items into which those Unex's V2X communication solution may be incorporated (collectively, the "Controlled Items"), may be subject to the export and foreign trade control laws and regulations of the United States[1] and the export, re-export, transfer, re-transfer, or transmission of these items ("export/re-export") therefore may be restricted by U.S. law. The export/import laws of other countries also may apply with respect to the Controlled Items.

1. You may not export/re-export Controlled Items to any country or person (individual, entity, or organization) to which such export/re-export is restricted or prohibited under U.S. Foreign Trade Control Laws or other applicable laws unless you have obtained all required licenses for such export/re-export. You have sole responsibility to obtain any license or other authorization required for you to export/re-export the Controlled Items.

2. You are expected to comply fully with all requests by Unex for information required in connection with the sale and shipment of the Controlled Items and with all terms and conditions of any export/import licenses or authorizations relating to the Controlled Items.

3. You may not use the Controlled Items in or in relation to (i) nuclear end uses; (ii) rocket/missile systems or unmanned air vehicles; (iii) nuclear, biological, or chemical weapons or weapons of mass destruction; (iv) non-U.S. maritime nuclear propulsion projects; (v) non-U.S. vessels or aircraft; (vi) any military end uses prohibited or restricted by U.S. Foreign Trade Control Laws, including 15 C.F.R. Part 744; or (vii) any other end use or end user prohibited by applicable law or terms of a license. You also will not export/re-export or transfer (in country) the Controlled Items for any of these restricted end uses or to any end users involved in these restricted end uses.

Important Information

1. Purpose:

The sales of Unex's Products are for the purposes of (i) evaluation and deployment of Unex's Products and (ii) software development of your V2X and related applications, and integrating or embedding Unex's Products into your products for V2X deployment (together, the "Purpose").

2. Restrictions:

The sale of Unex's Products does not convey any license expressly or by implication, except the limited license to use Unex's Products for the Purpose. You shall not license to any third party any part of the Unex's Products, except pursuant to the terms and conditions of a separate, written commercial supply and/or license agreement mutually agreed by Unex. You shall not, or allow any third party, to (i) reverse engineer, decompile, decrypt, unlock, reverse compile, disassemble, modify, create derivative works, or attempt to reconstruct or discover any ideas or algorithms of any Unex's Products, or (ii) use the Unex's Products for any purpose other than the Purpose. You will not remove, alter or obfuscate any patent markings on the Unex's Products or otherwise cause any patent on the Unex's Products to become ineffective under applicable patent laws.

3. Title:

Subject only to the express license herein to use Unex Products for the Purpose, as between you and Unex, Unex shall have all right, title and interest in and to the Unex's Products (and all improvements and modifications thereto) and all intellectual property rights therein. No licenses are granted hereunder by implication or estoppel.

4. Limited Warranties:

4.1: Unex's Products Warranties:

Unex warrants that each Unex's Products sold by Unex: (i) will substantially conform to the features specified in its then-current datasheet, as amended by Unex from time to time, for a period of ninety (90) days from the date of shipment by Unex; and (ii) will be free from defects in workmanship and materials under normal use and service for one (1) year from such shipment date. If a Unex product does not operate as warranted during its warranty period, and provided that you notify Unex promptly within the warranty period and comply with the Unex's RMA Procedures, Unex shall, in its sole discretion (a) return to Unex for repair in accordance with Unex's then-current Return Material Authorization ("RMA") procedures or such other procedures as are expressly agreed to in writing by an authorized representative of Unex. Unex shall have no responsibility or liability for returns not made in accordance with such Unex's RMA Procedures. (b) deliver to you the equivalent Unex's Products to replace such defective Unex's Products at its expense or (c) if replacement is not feasible, refund or issue a credit to you (at Unex's sole discretion) in the amount of the purchase price paid for the defective Unex's Products. Replacement of Unex's Products is warranted for ninety (90) days from shipment, or the balance of the original warranty period, whichever is longer.

4.2 Warranties Exclusive:

4.2.1 Sole Remedy:

THE REMEDIES STATED IN SECTION 4.1 ARE THE SOLE REMEDIES FOR ANY BREACHES OF THE WARRANTIES STATED IN THAT SECTION. TO THE FULLEST EXTENT ALLOWED BY LAW, THE WARRANTIES SET FORTH IN THIS AGREEMENT ARE EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES OR CONDITIONS, EXPRESS OR IMPLIED, EITHER IN FACT OR BY OPERATION OF LAW, STATUTORY OR OTHERWISE, INCLUDING BUT NOT LIMITED TO WARRANTIES, TERMS OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, CORRESPONDENCE WITH DESCRIPTION, NON-INFRINGEMENT, AND ACCURACY OF INFORMATION GENERATED, ALL OF WHICH ARE EXPRESSLY DISCLAIMED. UNEX'S WARRANTIES RUN ONLY TO YOU AND ARE NOT EXTENDED TO ANY THIRD PARTIES, WHICH FOR THE AVOIDANCE OF DOUBT, INCLUDES YOUR AGENTS, RESELLERS OF YOUR PRODUCTS AND END USERS. UNEX NEITHER ASSUMES NOR AUTHORIZES ANY OTHER PERSON TO ASSUME FOR IT ANY OTHER LIABILITY IN CONNECTION WITH THE SALE, INTEGRATION, MAINTENANCE OR USE OF THE UNEX'S PRODUCTS.

4.2.2 No Unex Defect:

UNEX'S WARRANTIES DO NOT APPLY TO A UNEX'S PRODUCTS IF IT DETERMINES THAT THE ALLEGED DEFECT IN THE UNEX'S PRODUCTS DOES NOT EXIST OR SUCH DEFECT WAS CAUSED BY YOUR, END USER'S OR ANY THIRD PARTY'S MISUSE, NEGLIGENCE, IMPROPER INSTALLATION OR TESTING, ATTEMPTS TO REPAIR OR USE BEYOND THE RANGE OF THE UNEX'S PRODUCTS INTENDED USE OR BY ACCIDENT, FIRE, LIGHTNING OR OTHER HAZARD OR ANY OTHER CAUSE BEYOND UNEX'S CONTROL.

4.3 Your Responsibilities:

You are solely responsible for the selection of the Unex's Products, their ability to achieve the results. Your intends to use with any hardware, software, peripherals or any system, and the performance that you, your customers and end users obtain from using them. You alone shall assume any and all warranty obligations with your customers and end users for each of your product that incorporates the Unex's Products, and you have no authority to obligate Unex in any way under each such warranty.

5. Liability and Disclaimer:

5.1 Limitation of Liability:

NOTWITHSTANDING ANY OTHER PROVISION OF AGREEMENT, IN NO EVENT SHALL UNEX (OR ITS SUPPLIERS, DISTRIBUTORS OR LICENSORS) BE LIABLE FOR ANY INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL OR PUNITIVE DAMAGES, OR FOR ANY LOSS OF PROFITS, REVENUE, BUSINESS OR DATA, ARISING FROM OR IN CONNECTION WITH THIS PURCHASE, OR THE SALE, INTEGRATION, USE, OR INABILITY TO USE, DELIVERY OR FAILURE TO DELIVER, PERFORMANCE, FAILURE OR INTERRUPTION OF PERFORMANCE OF THE UNEX'S PRODUCTS PROVIDED HEREUNDER, EVEN IF UNEX (OR ITS SUPPLIERS, DISTRIBUTORS OR LICENSORS, AS APPLICABLE) HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND REGARDLESS OF THE THEORY (WHETHER BASED IN CONTRACT, TORT, OR OTHERWISE) ON WHICH DAMAGES ARE SOUGHT. THESE DISCLAIMERS OF LIABILITY WILL APPLY EVEN IF THE REMEDIES PROVIDED HEREIN FAIL OF THEIR ESSENTIAL PURPOSE.

5.2 Special Applications:

The Unex's Products are not designed and are not warranted to be suitable for use in applications involving the risk of personal injury or death or the destruction of property, medical life support devices, or similar applications or any hazardous or high risk environments and activities requiring failsafe performance, such as the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems ("Special Applications"). Use of any Unex's Products in a Special Application without the express written approval of Unex is prohibited and UNEX SHALL NOT BE LIABLE FOR ANYTHING ASSOCIATED WITH SUCH USE. You assume any and all risks associated with the use of the Unex's Products in Special Applications and shall indemnify, defend, and hold Unex harmless from any third-party claims arising out of or related to the use of the Unex's Products in any such Special Applications.

6. Safety Regulations:

You shall ensure that Unex's Products are used in compliance with applicable vehicular and road safety laws and regulations. Unex does not provide any assurance that Unex's Products are compliant with applicable law in any specific jurisdiction.

Table of Contents

1. Introduction

Unex's OBU-201U contains V2X elements of DSRC/GNSS/HSM with pre-integrated IEEE 802.11p driver and IEEE 1609.x protocol stack (version 2016) running on ThreadX RTOS.

1.1 Acronyms

DSRC	Dedicated Short Range Communication
CCH	Control Channel
SCH	Service Channel
PSC	Provider Service Context
PSID	Provider Service Identifier
WAVE	Wireless Access in Vehicular Environments
WSA	WAVE Service Advertisements
WSMP	WAVE Short Message Protocol
WME	WAVE Management Entity
BSM	Basic Safety Message
SPAT	Signal Phase And Timing Message
MAP	Map Data
TIM	Traveler Information Message
RSA	Road Side Alert
RTCM	Radio Technical Commission For Maritime Services
CSR	Common Safety Request Message
NMEA	Common Safety Request Message

NMEA	Common Safety Request Message
NMEA	National Marine Electronics Association
ICA	Intersection Collision Alert Message
EVA	Emergency Vehicle Alert Message
PDM	Probe Data Message
SSM	Signal Status Message
SRM	Signal Request Message
PVD	Probe Vehicle Data Message
PSM	Pedestrian Safety Message

1.2 Power cable usage Note

For application flexibility in lab and vehicle, two kinds of power cables are provided. One is the round jack power adapter and the other one is 4-pin automotive power connector. It is recommended to use any ONE of the power cables at one time.

2. Configuration using CLI

2.1 GPS status:

To retrieve GPS status, we have: `gps show`

Example:

```
ate> gps show

GPSINFO: time: Thu Dec 8 02:41:56 2016

         latitude 24.7770760, longitude 121.0437980

         altitude 146.0200000 m

         ground speed 0.2057778 m/s

         angle 153.3000000

Total 10 records.

GPSINFO[0]: time: Thu Dec  8 02:41:56 2016

GPSINFO[1]: time: Thu Dec  8 02:41:56 2016

GPSINFO[2]: time: Thu Dec  8 02:41:56 2016

GPSINFO[3]: time: Thu Dec  8 02:41:56 2016

GPSINFO[4]: time: Thu Dec  8 02:41:56 2016

GPSINFO[5]: time: Thu Dec  8 02:41:56 2016

GPSINFO[6]: time: Thu Dec  8 02:41:56 2016

GPSINFO[7]: time: Thu Dec  8 02:41:56 2016
```

```
GPSINFO[8]: time: Thu Dec  8 02:41:56 2016

GPSINFO[9]: time: Thu Dec  8 02:41:55 2016
```

2.2 Initializing 1609.3 test

To initialize 1609.3 test, we have the following primitive: `dot3_test_init`

Syntax: `dot3_test_init`

Example: `dot3_test_init`

NOTE: For exit, we have, `dot3_test_exit`

2.3 Provider service – Creation/Deletion

To create a provider service, we have the following primitive:

`dot3_provider_serv_create`

Syntax: `dot3_provider_serv_create <handle> <psid> <repeat_rate> <radio_number> <service_channel> <channel_access> <priority> <is_secure> <sign_lifetime>`

The parameters are described in the following table:

Parameter	Valid Range	Description
handle	0 ~ 255	Handler id
psid	1 ~ 127	psid of desired service
repeat rate	0 ~ 255	repeat rate of WSA packets
radio_number	1 - 2	radio number of the service

service_channel	172 174 176 180 182 184	Channel number of desired service
channel_access	1: CCH 2: SCH 3: BOTH	The time slot for the desired channel
priority	0 ~ 15	Priority of user service. Lower value has higher priority
is_secure	0: no secured WSA 1: secured WSA	Security enabling status
sign_lifetime	10 ~ 30000 ms	life time of signature

Example: `dot3_provider_serv_create 1 123 1 1 172 0 5 0 10`

To delete a provider service, we have the following primitive:

```
dot3_provider_serv_delete
```

Syntax: `dot3_provider_serv_delete <handle> <serv_index>`

NOTE: The parameters <handle> and <serv_index> are assigned by the system and are not configurable by the user.

Example: `dot3_provider_serv_delete 1 0`

2.4 User service – Creation/Deletion

To create a user service, we have the following primitive:

```
dot3_user_serv_create
```

Syntax: *dot3_user_serv_create <handle> <psid> <request_type> <priority> <radio_number> <channel_number>*

The parameters are described in the following table:

Parameter	Valid Range	Description
handle	0 ~ 255	Handler id
psid	1 ~ 127	psid of desired service
request_type	0: MATCH 1: NO SCH Access	Access mode of user service
priority	0 ~ 15	Priority of user service. Lower value has higher priority
radio_number	1 - 2	radio number of the service
channel_number	172 174 176 180 182 184	Channel number of desired service

Example: *dot3_user_serv_create 1 123 0 5 2 172*

To delete a user service, we have the following primitive:

```
dot3_user_serv_delete
```

Syntax: *dot3_user_serv_delete <handle> <serv_index>*

NOTE: The parameters <handle> and <serv_index> are assigned by the system and are not configurable by the user.

Example: *dot3_user_serv_delete 1 0*

2.5 Channel service – Creation/Deletion

To create a channel service, we have the following primitive:

dot3_channel_serv_create

Syntax: *dot3_channel_serv_create <handle> <radio_number> <channel_number> <slot> <priority>*

The parameters are described in the following table:

Parameter	Valid Range	Description
handle	0 ~ 255	Handler id
radio_number	1 - 2	the radio number
channel_number	172 174 176 178 180 182 184	The channel that users want to reserve
slot	1: CCH 2: SCH 3: BOTH	The time slot for the desired channel
priority	0 ~ 15	Priority of user service. Lower value has higher priority

Example: *dot3_channel_serv_create 1 1 172 3 5*

To delete a channel service, we have the following primitive:

dot3_channel_serv_delete

Syntax: *dot3_channel_serv_delete <handle> <serv_index>*

NOTE: The parameters <handle> and <serv_index> are assigned by the system and are not configurable by the user.

Example: *dot3_channel_serv_delete 1 0*

2.6 WSM service – Creation/Deletion

To create a WSM service, we have the following primitive:

dot3_wsm_serv_create

Syntax: *dot3_wsm_serv_create <handle> <psid>*

The parameters are described in the following table:

Parameter	Valid Range	Description
handle	0 ~ 255	Handler id
psid	1 ~ 127	psid of desired service

Example: *dot3_wsm_serv_create 1 123*

To delete a wsm service, we have the following primitive:

dot3_wsm_serv_delete

Syntax: *dot3_wsm_serv_delete <handle> <serv_index>*

NOTE: The parameters <handle> and <serv_index> are assigned by the system and are not configurable by the user.

Example: *dot3_wsm_serv_delete 1 0*

2.7 Checking MIB status

To retrieve MIB setting, we have the following primitive: `dot3_show_mib`

Illustration:

Initialize 1609.3 test and create a channel service

```
ate> dot3_test_init

wme_init success, handle = 1



ate> dot3_channel_serv_create 1 1 172 3 5

index = 0, channel 178 BOTH delete

dev_id = 0, channel = 170

index = 0, channel 172 CCH add

dev_id = 1, channel = 172

Channel 172 assigned ...

wme_channel_serv_create success, serv_index = 0
```

View MIB status:

```
ate> dot3_show_mib

wme_init success, handle = 2

*** Provider entry ***

*** User entry ***

*** Channel entry ***
```

```
index = 0

priority  = 5

channel  = 172

slot  = 3


*** Wsm entry ***

*** Available entry ***

ate>
```

As shown above, we can see a Channel entry in MIB.

2.8 Security Test command (1609 dot2)

To test security, we have the following primitive: `test_us_sec`

Example: `test_us_sec 125 1 1 255 1`

2.9 BSM Tx & Rx

To send a BSM packet, we have the following primitive: `txBsm`

Syntax:
```
txBsm <psid> <data_rate> <tx_power> <dest_mac> <interval>
      <num>
```

The parameters are described in the following table:

Parameter	Valid Range	Description
psid	1 ~ 127	psid of desired service
data_rate	6 9 12 18 24 36 48 54	data rate of sending WSM packets
tx_power	12- 25, unit:dBm	tx power level of sending WSM packets
dest_mac	format: XX:XX:XX:XX:XX:XX	peer mac address of sending WSM packets
interval	0 - 1000, unit:ms	the interval of sending packets
num	1 - 10000	the number of sending packets

Example: `txBsm 123 6 18 FF:FF:FF:FF:FF:FF 1000 1`

To receive BSM packets, we have the following primitive: `rxBsm`

Syntax: `rxBsm <psid>`

Example: `rxBsm 123`

NOTE: Syntax is the same for other J2735 message sets which are:

SPAT/MAP/TIM/RSA/RTCM/CSR/NMEA/ICA/EVA/PDM/SSM/SRM/PVD/PSM.

2.10 Firmware Upgrade

1. Setup a serial connection to the unit

2. Ensure that the unit is connected to an Ethernet LAN

3. Reset the unit and enter U-Boot console by pressing any key during the 3 second countdown. You should see the U-Boot console prompt:

   ```
   U-Boot>
   ```

4. We will assume the IP address of your TFTP server is 10.10.10.10. Perform ping test

   ```
   U-Boot> ping 10.10.10.10
   ```

5. If the ping was successful, you should see output similar to:

   ```
   Link: UP

   Duplex: FULL

   Speed 100BASE-X

   Using device

   host 10.10.0.10 is a   live
   ```

6. Configure the TFTP server's IP address:

   ```
   U-Boot> setenv serverip 10.10.10.10
   ```

7. We will assume that the image you would like to boot is served by the TFTP server under the name uImage. Issue the following command to load the image via TFTP:

   ```
   U-Boot> tftp uImage
   ```

8. Erasing flash memory is done differently depending on flash size. When using a 8 MB flash device:

```
U-Boot> protect off 80000 7fffff
U-Boot> erase 80000 7fffff
```

9. When using a 32 MB flash device:

```
U-Boot> mw.l 0x41016850 0x001f0000
U-Boot> protect off 80000 1fdffff
U-Boot> erase 80000 1fdffff
```

10. Replace existing firmware image with the new one:

```
U-Boot> cp.b ${fileaddr} 80000 ${filesize}
U-Boot> setenv bootcmd cp.b 80000 50000000 ${filesize}\; bootm
U-Boot> saveenv
```

11. You should see output similar to:

```
U-Boot> tftp uImage
Link: UP
Duplex: FULL
Speed 100BASE-X
Using device
TFTP from server 10.10.0.10; our IP address is 10.10.0.50
Filename ’uImage’.
Load address: 0x50000000
```

```
Loading: TftpRemotePort=69

#################################################################

###########################################

done

Bytes transferred = 552100 (86ca4 hex)

U-Boot> protect off 80000 7fffff

..................................................................
done

Un-Protected 120 sectors

U-Boot> erase 80000 7fffff

..................................................................
done

Erased 120 sectors

U-Boot> cp.b ${fileaddr} 80000 ${filesize}

Copy to Flash... done

U-Boot> setenv bootcmd cp.b 80000 50000000 ${filesize}\; bootm

U-Boot> saveenv

Saving Environment to Flash...

.. done

Un-Protected 2 sectors

Erasing Flash...

.. done
```

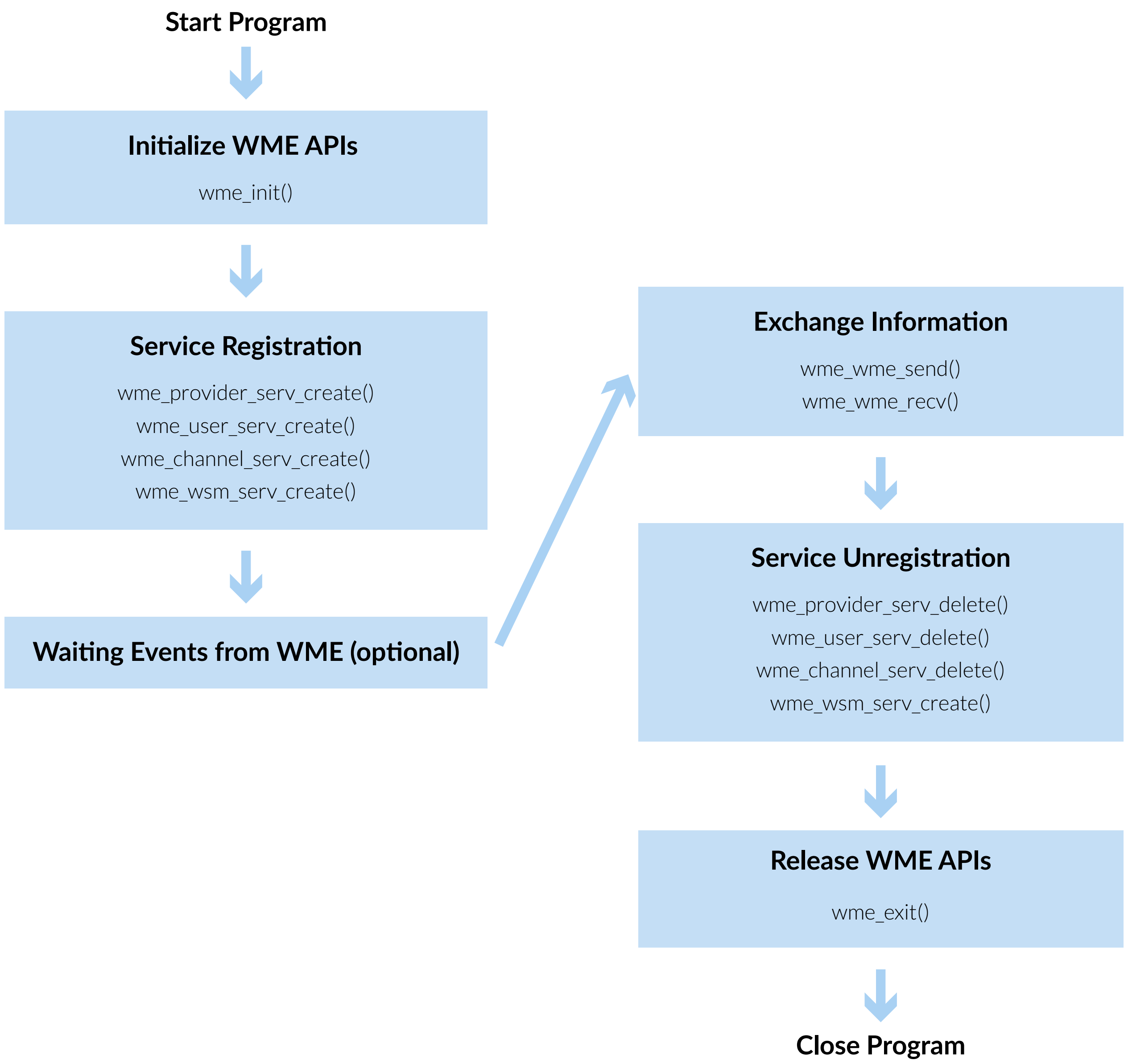
```
Erased 2 sectors

Writing to Flash... done

.. done

Protected 2 sectors

Done
```

12. Reset the unit via the command:

```
U-Boot> reset
```

3. IEEE 1609.3 PROGRAMMING API

The process for using IEEE 1609.3 networking services API is shown below:



3.1 Initialize WME APIs

```
int wme_init(wme_handle_t *handle);
```

This function will initialize the WME service and get at new handle id for the service. Further, the handle id can be used to create or delete provider services, user services, channel services or wsm services.

- On success, zero is returned.

- On error, a negative value is returned.

The function's parameters are described in the following table.

Parameter	Description
handle	A new handle id for WME service

3.2 Provider Service Creation/Deletion

The WME system provides four kinds of services: Provider, User, Channel and WSM.

Provider service APIs are as follows:

```
int     wme_provider_serv_create (wme_handle_t *handle, unsigned
        short *serv_index, wme_provider_info_t *info);

int     wme_provider_serv_change (wme_handle_t *handle, unsigned
        short serv_index, wme_provider_info_t *info);

int     wme_provider_serv_delete (wme_handle_t *handle, unsigned
        short serv_index);
```

A provider service can be created, changed, or deleted by using the above mentioned APIs. For creation, the function returns a new service index to application users. When calling the function, the system will assign a specific channel if the channel is available. If success, the system will generate a corresponding WSA, and send the WSA periodically. For change or deletion, the function needs a valid service index to change or delete the service. All the three functions also need a valid handle id which is obtained from the function *'wme_init()'.*

- On success, zero is returned.

- On error, a negative value is returned.

The function's parameters are described in the following table.

Parameter	Description
handle	A valid handle id for a WME service
serv_index	For creation, a new service index for the provider service For change or deletion, a valid service index for the provider service
info	Parameters of a provider service

Structure of the provider information is defined as below:

```
typedef struct provider_info

{

        unsigned char       dest_mac[MAC_SIZE];

        unsigned char       wsa_type;

        unsigned int        psid;

        unsigned char       psc[PSC_SIZE];

        unsigned char       psc_len;

        unsigned char       service_priority;

        unsigned char       radio_number;
```

```
            unsigned char       service_channel;
            unsigned char       wsa_channel;
            unsigned char       channel_access;
            unsigned char       repeat_rate;
            unsigned char       ip_service;
            unsigned char       ipv6_addr[IPV6_SIZE];
            unsigned short      service_port;
            unsigned char       provider_mac[MAC_SIZE];
            unsigned char       rcpi_threshold;
            unsigned char       wsa_count_threshold;
            unsigned char       wsa_count_threshold_interval;
            unsigned int        info_element_indicator;
            unsigned short      certificate_id;
            unsigned short      signature_lifetime;
      } wme_provider_info_t;
```

The fields are described in the following table.

Parameter	Description
dest_mac	The destination MAC address of WSA. Not supported currently.
wsa_type	Valid value: WSA_SCURED, WSA_UNSCURED
psid	The desired Provider Service Identifier (PSID)

psc	The description is associated to the psid, and the max length is PSC_SIZE (31 bytes).
psc_len	The real length of PSC
service_priority	Service priority: 0 ~ 15. The lower value has higher priority.
radio_number	Valid value: RADIO_0 or RADIO_1
service_channel	Valid value: 172, 174, 176, 180, 182, 184. The control channel can't be used.
wsa_channel	NO Support, only 178 now
channel_access	Valid value: SLOT_BOTH, SLOT_SCH, SLOT_CCH
repeats	The number of WSAs to be sent per 5 seconds. The real sending rate is equal to repeats +1. Valid value: 0~ 255
ip_service	Set to 0. The IP service is not available now.
provider_mac	Specify the provider's mac address. It is optional. If no use, Set all the value to 0.
rcpi_threshold	The threshold of RCPI of received WSAs. Not supported currently.
wsa_count_threshold	It indicates the recommended minimum number of received WSAs. It is optional. If no use, set it to 0. Valid value: 0 ~ 255.
wsa_count_threshold_interval	It indicates the time interval over which received WSAs are counted. The unit is 100ms. It is optional. If no use, set it to 0. Then the remote user accepting the WSA will use the default interval (one second). Valid value: 0 ~ 255
info_element_indicator	It indicates which optional information can be inserted into WSAs. Valid value: BITMASK_PSC, BITMASK_PROVIDER_MAC, BITMASK_RCPI_THRESHOLD, BITMASK_WSA_COUNT_THRESHOLD, BITMASK_WSA_COUNT_THRESHOLD_INTERVAL
certificate_id	Id of the certificate for signing WSAs. Only for secured WSAs
signature_lifetime	The number of milliseconds over which the WSA signature should be valid. Only for secured WSAs Valid value: 10 ~ 30000

3.3 User Service Creation/Deletion

User service APIs are as follows:

```
int     wme_user_serv_create (wme_handle_t *handle, unsigned short
        *serv_index, wme_user_info_t *info);

int     wme_user_serv_delete (wme_handle_t *handle, unsigned short
        serv_index);
```

A user service is created or deleted by using the above mentioned APIs. For creation, the function returns a new service index to application users. When calling the function, the system will try to match the received WSAs. If the service matches a received WSA and the priority is higher, the system will assign a specific channel for the service. For deletion, the function will need a valid service index to delete the service. All the two functions also need a valid handle id which is obtained from the function `'wme_init()'.`

- On success, zero is returned.

- On error, a negative value is returned.

The function's parameters are described in the following table.

Parameter	Description
handle	A valid handle id for a WME service
serv_index	For creation, a new service index for the provider service For change or deletion, a valid service index for the provider service
info	Parameters of a user service

The structure of the user information is defined as:

```
    typedef struct user_info

    {

            unsigned char         request_type;
```

```
        unsigned int          psid;
        unsigned char         psc[PSC_SIZE];
        unsigned char         psc_len;
        unsigned char         service_priority;
        unsigned char         wsa_type;
        unsigned char         radio_number;
        unsigned char         channel_number;
        unsigned char         src_mac[MAC_SIZE];
        unsigned char         advertiser_id[ADVERTISER_ID_SIZE];
        unsigned char         advertiser_id_len;
        unsigned char         link_quality;
        unsigned char         immediate_access;
   } wme_user_info_t;
```

The fields are described in the following table:

Field	Description	Match
request_type	Valid value: ACCESS_ON_MATCH, ACCESS_NO_SCH	
wsa_type	Valid value: WSA_UNSECURE, WSA_SCURED, WSA_BOTH, WSA_ANY	
psid	The receiving Provider Service Identifier (PSID)	•
psc	The description is associated to the psid, and the max length is PSC_SIZE (31 bytes).	•
psc_len	The real length of PSC	
service_priority	Service priority: 0 ~ 15. The lower value has higher priority.	

radio_number	Valid value: RADIO_0 or RADIO_1	
channel_ number	The user service will match up the provider service with the channel. Valid value: 172, 174, 176, 180, 182, 184. The control channel can’t be used.	•
src_mac	The user will match up the provider service with the MAC. The value 1 in all bits indicates that any MAC is accepted.	•
advertiser_id	The description is associated to the device sending WSAs, and the max length is ADVERTISER_ID_SIZE (32 bytes).	
advertiser_id _len	The real length of the advertiser_id	
link_quality	The threshold of the quality for received WSAs. Not supported currently.	
immediate_ access	Only support 0 (Channel switch) or 255 (Continuous)	

3.4 Channel Service Creation/Deletion

Channel service APIs are as follows:

```
int     wme_channel_serv_create(wme_handle_t *handle, unsigned short
        *serv_index, wme_user_info_t *info);

int     wme_channel_serv_delete(wme_handle_t *handle, unsigned short
        serv_index);
```

A channel service is created or deleted by using the above mentioned APIs. For creation, the function returns a new service index to application users. When calling the function, the system will try to assign a specific channel if the priority of the service is higher. For deletion, the function needs a valid service index to delete the service. Both the functions need a valid handle id which is obtained from the function ‘wme_init()’.

- On success, zero is returned.

- On error, a negative value is returned.

The function's parameters are described in the following table.

Parameter	Description
handle	A valid handle id for a WME service
serv_index	For creation, a new service index for the provider service For change or deletion, a valid service index for the provider service
info	Parameters of a channel service

The structure of the channel information is defined as:

```
typedef struct channel_info

{

      unsigned char       service_priority;

      unsigned char       radio_number;

      unsigned char       channel_number;

      unsigned char       time_slot;

} wme_channel_info_t;
```

The fields are described in the following table:

Field	Description
service_priority	Service priority: 0 ~ 15. The lower value has higher priority.
radio_number	Valid value: RADIO_0 or RADIO_1
channel_number	The channel that users desired to reserve Valid value: 172, 174, 176, 178, 180, 182, 184.
time_slot	The time slot for the desired channel.

3.5 WSM Service Creation/Deletion

To create/delete a WSM service, we have the following primitives:

```
int     wme_wsm_serv_create (wme_handle_t *handle, unsigned short
        *serv_index, wme_user_info_t *info);

int     wme_wsm_serv_delete (wme_handle_t *handle, unsigned short
        serv_index);
```

A wsm service is created or deleted by using the above mentioned APIs. For creation, the function returns a new service index to application users. When calling the function, the system will receive wsm packets with the specific PSID. For deletion, the functions need a valid service index to delete the service. Both the functions need a valid handle id which is obtained from the function ‘wme_init()’.

- On success, zero is returned.

- On error, a negative value is returned.

The function's parameters are described in the following table.

Parameter	Description
handle	A valid handle id for a WME service
serv_index	For creation, a new service index for the wsm service For change or deletion, a valid service index for the wsm service
info	Parameters of a wsm service

The structure of the channel information is defined as:

```
typedef struct wsm_info

{

    unsigned int psid;

} wme_wsm_info_t;
```

The fields are described in the following table:

Field	Description
psid	The receiving Provider Service Identifier (PSID)

3.6 Sending WSM packets

For sending WSM packets, we have the following primitive:

```
int wme_wsm_send (wme_handle_t *handle, struct out_wsm *wsm);
```

Users can use the function to send WSM packets on a specific channel. The function needs a valid handle id which is obtained from the function *'wme_init()'.*

- On success, the transmitted packet size is returned.

- On error, a negative value is returned.

The function's parameters are described in the following table.

Parameter	Description
handle	A valid handle id for a wme service
wsm	Parameters of a wsm packet

The structure for out_wsm is defined as:

```
typedef struct out_wsm

{

      unsigned char       radio_number;

      unsigned char       channel_number;

      unsigned char       data_rate;

      signed char         txpwr_level;

      unsigned char       user_priority;

      unsigned int        psid;

      unsigned long long  expiry_time;  /* Not supported now */

      unsigned char       dest_mac[MAC_SIZE];

      struct {

            unsigned char   channel:1,

                             rate:1,

                             power:1,

                             load:1,

                             reserved:4;
            } extensions;

            unsigned char        data[WSM_MAX_SIZE];

            unsigned short       length;

        } wme_out_wsm_t;
```

The fields are described in the following table:

Field	Description
radio_number	Valid value: RADIO_0 or RADIO_1
channel_number	Valid value: 172, 174, 176, 178, 180, 182, 184.
data_rate	The data rate in the packet Valid value: 6, 9, 12, 18, 24, 36, 48, 54
txpwr_level	The tx power in the sending packet Valid value: 0 ~ 20
user_priority	The priority in the sending packet. The value maps to corresponding EDCA. Valid value: 0 ~ 7
psid	Provider Service Identifier (PSID) Valid value: defined as 1609.3
dest_mac	The peer's MAC address
extensions	It indicates which of the WSMP header extension fields should be included in the packet.
data	The payload of the sending wsm pakcet
length	The length of the sending wsm pakcet

3.7 Receiving WSM packets

For receiving WSM packets, we have the following primitive:

```
int wme_wsm_recv (wme_handle_t *handle, struct in_wsm *wsm,
    unsigned int timeout);
```

Users can use the function to receive WSM packets after registering the wsm service with the desired psid. The function needs a valid handle id which is obtained from the function `'wme_init()'.`

- On success, zero is returned.

- On error, a negative value is returned.

The function parameters are described in the following table.

Parameter	Description
handle	A valid handle id for a wme service
wsm	Parameters of a received wsm packet
timeout	The timout of waiting to receive packets

The structure for in_wsm is defined as:

```
typedef struct in_wsm

{

  unsigned char        version;

  unsigned char        radio_number;

  unsigned char        channel_number;

  unsigned char        data_rate;
  signed char         txpwr_level;
  unsigned char       user_priority;  /* Not supported now */
  unsigned int        psid;
  unsigned char       src_mac[MAC_SIZE];
  struct {
     unsigned char    channel:1,
                      rate:1,
                      power:1,
                      load:1,
                      reserved:4;
  } extensions;
  signed char         rssi;
  unsigned char       real_channel;
  unsigned char       data[WSM_MAX_SIZE];
  unsigned short      length;
}wme_in_wsm_t;
```

The fields are described in the following table:

Field	Description
version	The version of WSM
radio_number	Not supported currently.
channel_number	The channel number in the extension field of the received wsm packet Valid value: 172, 174, 176, 178, 180, 182, 184.
data_rate	The data rate in the extension field of the received wsm packet Valid value: 6, 9, 12, 18, 24, 36, 48, 54
txpwr_level	The tx power in the extension field of the received wsm packet
user_priority	The priority in the received packet. The value maps to corresponding EDCA. Valid value: 0 ~ 7 Not supported currently.
psid	Provider Service Identifier (PSID) Valid value: defined as 1609.3
src_mac	The peer's MAC address
extensions	It indicates which of the WSMP header extension fields should be included in the packet.
rssi	The rssi of the received wsm packet
real_channel	The real channel number of the received wsm packet
data	The payload of the received wsm pakcet
length	The length of the received wsm pakcet

3.8 Event waiting

For event waiting, we have the following primitive:

```
int wme_event_recv(wme_handle_t *handle, struct event_message
     *event, unsigned int timeout);
```

If a service is registered, the system may notify users of available channels or upcoming matching services. The following function can be called to wait events from the wme system.

- On success, zero is returned.

- On error, a negative value is returned.

The function's parameters are described in the following table.

Parameter	Description
handle	A valid handle for a wme service
event	The received event from the wme system
timeout	The timout of waiting to receive events

The structure for event_message is defined as:

```
typedef struct event_message

{

        unsigned char   event;

        unsigned char   reason;

        union {

                struct     event_channel channel;

                struct mib_available_service_info service;
```

```
            } info;

        } wme_event_t;
```

The fields are described in the following table:

Field	Description
event	Valid value: EVENT_SERVICE, EVENT_CHANNEL,EVENT_MATCH
reason	Valid value: REASON_SERVICE_AVAILABLE, REASON_SERVICE_UNAVAILABLE, REASON_CHANNEL_AVAILABLE, REASON_CHANNEL_UNAVAILABLE
info	The additional information of the event

3.9 Getting MIB Information

To get MIB information, we have the following primitive:

```
int wme_mib_get(struct mib_info *info);
```

Users can use the above function to get information of the system.

- On success, zero is returned.

- On error, a negative value is returned.

The function parameters are described in the following table.

Parameter	Description
handle	A valid handle id for a wme service
info	The related information of the queried mibs

The structure for mib_info is defined as:

```
typedef struct mib_info

{

    unsigned char       entry_type;

    unsigned char       next;

    short               entry_index;

    union {

        wme_provider_serv_mib_t    provider_entry;

        wme_user_serv_mib_t         user_entry;

        wme_channel_serv_mib_t      channel_entry;

        wme_wsm_serv_mib_t          wsm_entry;

        wme_available_serv_mib_t    available_entry;

    } entry_value;

} wme_mib_entry_t;
```

The fields are described in the following table:

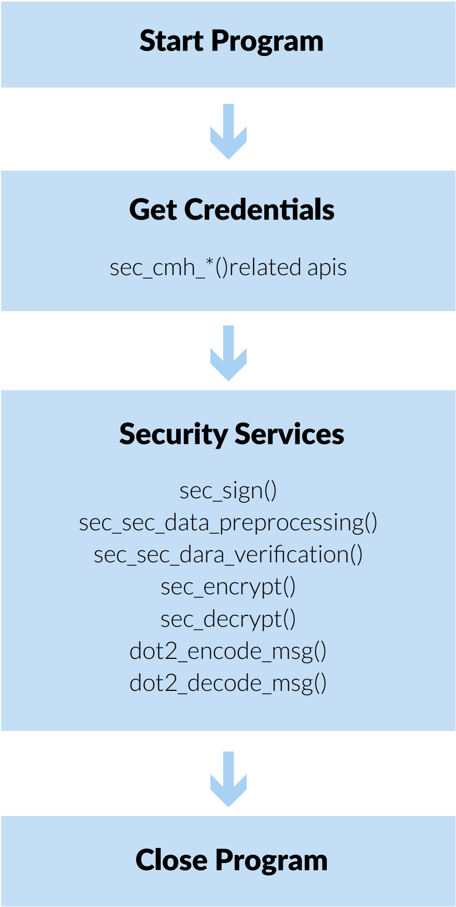
Field	Description
entry_type	Valid value: PROVIDER_ENTRY, USER_ENTRY, CHANNEL_ENTRY, WSM_ENTRY, AVAILABLE_ENTRY
next	Valid value: 0, 1 1: fetch the next available index 0: fetch the index
entry_index	The index of a specific mib entry
entry_value	The value of a specific mib entry

3.10 Example code

Please refer to cmd_dot3_v3.c.

4. IEEE 1609.2 PROGRAMMING API

The process for using IEEE 1609.2 security service API is shown in figure below:



Start Program

Get Credentials

sec_cmh_*()related apis

Security Services

sec_sign()
sec_sec_data_preprocessing()
sec_sec_dara_verification()
sec_encrypt()
sec_decrypt()
dot2_encode_msg()
dot2_decode_msg()

Close Program

There are two parts of the APIs: (1) security credential related and (2) security service related.

(1) Security credential related APIs

For signing/verifying/encrypting/decrypting secure messages, the application would need corresponding certificates and corresponding keys. There are two ways to get the certificate and keys: load existing ones and generate/request new ones. The protocol stack supports APIs needed for implementing both by user for getting the security credentials. The details of these APIs are described in Section 4.1.

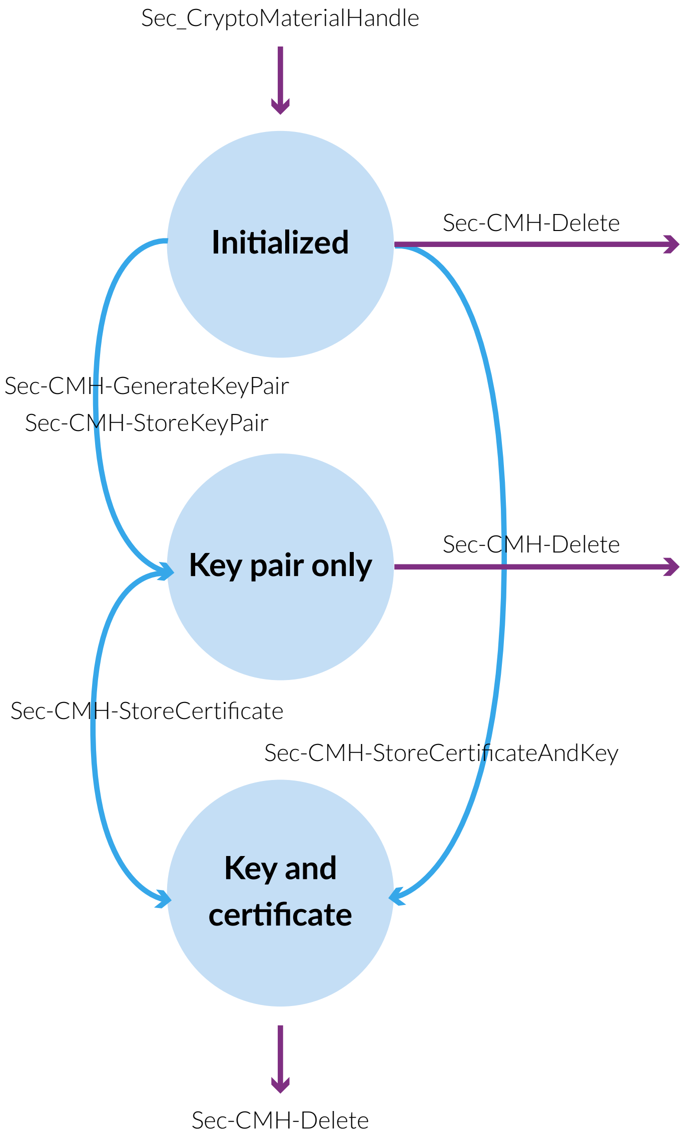
(2) Security service related APIs

These APIs are used to encode/decode secure messages and the details are described in Section 4.2.

All the enumerate values and return codes are defined in the header file dot2_api.h.

4.1 IEEE 1609.2 Security Credential Management

The protocol stack provides the APIs to manage security credentials according to the IEEE 1609.2 standard Section 9.2.2. The user can use these APIs to implement the desired scheme based on their applications. The following figure describes the transitions between each state:



According to the above state diagram, the corresponding primitives are listed as follows:

```
int     sec_cmh_new(unsigned int* o_cmh);

int     sec_cmh_gen_key_pair(unsigned int i_cmh, dot2_ec_algorithms_t i_alg, private_key_type_t i_key_type, unsigned char* o_pub_key);

int     sec_cmh_store_key_pair(unsigned int i_cmh, dot2_ec_algorithms_t i_alg, unsigned char* i_pub_key, unsigned char* i_pri_key);

int     sec_cmh_store_cert(unsigned int i_cmh, unsigned int i_cert_len, unsigned char* i_cert, unsigned char* i_pri_key_material);

int     sec_cmh_store_cert_and_key(unsigned int i_cmh, unsigned int i_cert_len, unsigned char* i_cert, unsigned char* i_pri_key);

int     sec_cmh_del(unsigned int i_cmh);
```

Following table shows the descriptions of these primitives.

Name of Primitive	Parameters	Description
sec_cmh_new	o_cmh: the cryptomaterial handle created	This primitive creates a new cryptomaterial handle and set into the initialized state.
sec_cmh_gen_key_pair	i_cmh: the cryptomaterial handle that would be binded with the generated key pair i_alg: the elliptic curve and algorithm used for generating the key pair i_key_type: the usage of the private key o_pub_key: the generated public key	The primitive would generate a key pair according to the input algorithm, key type and transform the referenced CMH to key pair only state. NOTE: If the key type was set to RECONSTRUCT_SEED, it may need to call sec_cmh_store_cert() primitive to reconstruct the actual private key. This is used to support acquiring certificate and keys from a PKI such as SCMS scheme.

sec_cmh_store_key_pair	i_cmh: the cryptomaterial handle that would be binded with the stored key pair i_alg: the elliptic curve and algorithm used for the imported key pair i_pub_key: the public key to be stored i_pri_key: the private key to be stored	The primitive would store the input key pair according to the input algorithm and transform the referenced CMH to key pair only state.
sec_cmh_store_cert	i_cmh: the cryptomaterial handle that would be binded with the stored certificate i_cert_len: the length in bytes of the certificate to be imported i_cert: the string of the certificate to be imported i_pri_key_material: the private reconstruction value to be used to reconstruct the actual private key	The primitive would store and bind the input certificate with the input CMH. The parameter i_pri_key_material would be ignored if the certificate is explicit type and the corresponding private key would be the one previously generated or stored by calling sec_cmh_gen_key_pair() or sec_cmh_store_key_pair(). If the certificate is implicit type, the security module would try to reconstruct the private key using the i_pri_key_material and the seed generated previously by calling sec_cmh_gen_key_pair().
sec_cmh_store_cert_and_key	i_cmh: the cryptomaterial handle that would be binded with the stored certificate and key i_cert_len: the length in bytes of the certificate to be imported i_cert: the string of the certificate to be imported i_pri_key: the private key to be stored	The primitive would store the certificate and key and transform the referenced CMH to key and certificate state.
sec_cmh_del	i_cmh: the cryptomaterial handle that would be deleted	The primitive would delete the referenced cryptomaterial handle and related data.

The protocol stack provides the APIs to manage symmetric cryptomaterial handles according to the IEEE 1609.2 standard section 9.2.3. The related primitives are listed as follows:

```
int     sec_scmh_new(dot2_sym_algorithms_t i_alg, unsigned char i_
        gen_flag, unsigned char* i_sym_key, unsigned int* o_scmh);
int     sec_scmh_hashid8(unsigned int i_scmh, unsigned char* o_
        hashid8);
int     sec_scmh_del(unsigned int i_scmh);
```

Following table shows the descriptions of these primitives:

Name of Primitive	Parameters	Description
sec_scmh_new	i_alg: the symmetric algorithm used with the symmetric key i_gen_flag: the flag to generate a symmetric key when allocate a new scmh or not i_sym_key: if i_gen_flag == 0, then importing the i_sym_key as the symmetric key o_scmh: the created symmetric cryptomaterial handle	This primitive creates a newly symmetric cryptomaterial handle and sets the related symmetric key.
sec_scmh_hashid8	i_scmh: the symmetric cryptomaterial handle to get the related hash value of the corresponding symmetric key o_hashid8: the least significant 8 bytes of the hash value of the symmetric key	The primitive would hash the symmetric key binded with the input SCMH and set the value to the output parameter o_hashid8.
sec_scmh_del	i_scmh: the symmetric cryptomaterial handle that would be deleted	i_scmh: the symmetric cryptomaterial handle that would be deleted

The protocol stack also provides an API for importing CA certificates. The added certificates would be recognized as trust anchors when verifying the secure messages. The API is listed below:

```
int     sec_cmh_store_ca_cert(unsigned int i_cert_len, unsigned
        char* i_cert);
```

The input parameters are the length in bytes and the content of the certificate.

NOTE: All the unsigned char strings above are the binary data of the content, not ASCII string.

4.1.1 Certificate Generation

For possible testing needs, the stack provides another API that can generate a certificate and related key with most fields set as default values.

```
int     dot2_generate_cert(unsigned int i_psid, unsigned int
        *o_cmh, unsigned int* o_cert_len, unsigned char* o_
        cert_str);
```

The parameter i_psid is the permitted PSID of the generated certificate. The primitive would allocate a CMH, generate a key pair and bind with the generated certificate. The generated materials would be set to the parameters o_cmh, o_cert_len, and o_cert_str.

4.2 IEEE 1609.2 Secure Messages

The following primitives provide IEEE 1609.2 secure services.

```
int     sec_sign(sec_sign_t* sign_info);

int     sec_sec_data_preprocessing(sec_sec_data_preprocessing_t*
        sec_data);

int     sec_sec_data_verification(sec_sec_data_verification_t* sec_
        data);

int     sec_encrypt(sec_encrypt_t* encrypt_info);

int     sec_decrypt(sec_decrypt_t* decrypt_info);
```

The structure of the parameter sign_info in sec_sign is defined as following:

```
typedef struct

{

    unsigned int        cmh;

    unsigned int        data_len;

    unsigned char*      data;

    dot2_data_type_t    data_type;

    unsigned char       ext_data_hash[32];

    unsigned char       ext_data_hash_alg;

    unsigned int        psid;

    unsigned char       set_gen_time;

    unsigned char       set_gen_loc;

    unsigned int        exp_time;
```

```
         dot2_signer_type_t    signer_id_type;

         char                  signer_id_cert_chain_len;

         unsigned char         max_cert_chain_len;

         dot2_ec_point_choice_t sign_with_fast_ver;

         dot2_ec_point_choice_t ecpoint_format;

         unsigned char         use_p2p_cert_dist;

         unsigned int          sdee_id;

         unsigned int          signed_data_len;

         unsigned char*   signed_data;

    } sec_sign_t;
```

And the fields are defined as following table.

Field	Description
cmh	The primitive would use the certificate and key binded with input cmh to sign the message.
data_len	The length in bytes of the input data.
data	The data to be signed.
data_type	The type should be UNSECURE or SIGNED. If the data type is UNSECURE, then the primitive would pack the input data into an Ieee1609Dot2Data structure before encoding it into a signed message.
ext_data_hash	Reserved, currently not used.
ext_data_hash_alg	Reserved, currently not used.
psid	The psid of the application which uses the 1609.2 secure messages. The encoding of psid would follow the encoding rules defined in IEEE 1609.3.

set_gen_time	The value of the parameter should be 0 or 1. It indicates the primitive would encode the generation time field or not.
set_gen_loc	The value of the parameter should be 0 or 1. It indicates the primitive would encode the generation location field or not.
exp_time	If the value is not zero, then the primitive would encode the expiration time field in the header and the expiration time would be set to the current time + the amount of exp_time (unit: ms).
signer_id_type	The value of the parameter should be DIGEST or CERTIFICATE. It indicates the signer identifier type of the signed message.
signer_id_cert_chain_len	Reserved, currently only encode the signer certificate.
max_cert_chain_len	Reserved, the value of the max_cert_chain_len should be larger than signer_id_cert_chain_len.
sign_with_fast_ver	The value of the parameter should be one of X_ONLY, COMPRESSED, and UN COMPRESSED. It indicates the primitive would encode the R value of the signature field with x-coordinate only, compressed point or uncompressed point type.
ecpoint_format	Reserved, currently not used.
use_p2p_cert_dist	Reserved, currently not used.
sdee_id	Reserved, currently not used.
signed_data_len	The length in bytes of the encoded signed message. NOTE: when calling the primitive, the signed_data_len should be set to the value of the maximum length of signed_data. If the encoded length is larger than the input value, it would fail to encode the signed message. If the encoding process succeeded, the signed_data_len would be set to the actual encoded length of the signed data.
signed_data	The encoded signed message. NOTE: the user should allocate enough space for the security module to fill the signed message before calling the primitive.

The structure of the parameter sec_data in sec_sec_data_preprocessing is defined as following:

```
typedef struct
{
     unsigned int          data_len;
     unsigned char*        data;
     unsigned int          sdee_id;
     unsigned char         use_p2p_cert_dist;

     dot2_data_type_t      content_type;
     unsigned int          psid;
     unsigned int          ssp_len;
     unsigned char         ssp[32];
     unsigned char         assurance_level;
     unsigned int          next_crl_time;

     unsigned int          raw_data_len;
     unsigned char*        raw_data;
     sec_sec_data_verification_t data_info;
} sec_sec_data_preprocessing_t;
```

And the fields are defined as following table.

Field	Description
data_len	The length in bytes of the input data.
data	The data to be decoded.
sdee_id	Reserved, currently not used.
use_p2p_cert_dist	Reserved, currently not used.
content_type	The extracted data type of the input data.
psid	The extracted PSID of the input data from the header field.
ssp_len	The corresponding ssp information related to the permitted PSID of the signer certificate.
ssp	The corresponding ssp information related to the permitted PSID of the signer certificate.
assurance_level	Reserved, currently not used.
next_crl_time	Reserved, currently not used.
raw_data_len	The length in bytes of the decoded data. NOTE: when calling the primitive, the raw_data_len should be set to the value of the maximum length of raw_data. If the extracted data length is larger than the input value, it would fail to decode the secure message. If the decoding process succeeded, the raw_data_len would be set to the actual decoded length of the secure data.
raw_data	The extracted data. NOTE: the user should allocate enough space for the security module to fill the extracted message before calling the primitive.
data_info	The structure may be used later for verification. The primitive would fill the extracted values from the header fields of the secure message.

The structure of the parameter sec_data in sec_sec_data_verification is defined as following:

```
typedef struct

{

        unsigned int                sdee_id;

        unsigned int                psid;

        dot2_data_type_t            content_type

        unsigned int                signed_data_len;

        unsigned char*              signed_data;

        unsigned char               external_data_hash[32];

        unsigned char               hash_alg;

        unsigned char               max_cert_chain_len;

        unsigned char               data_hash_for_replay_check[32];

        unsigned char               replay_check;

        unsigned long long          gen_time;

        unsigned char               gen_time_in_past_check;

        unsigned int                validity_period;

        unsigned char               gen_time_in_future_check;

        unsigned int                future_data_period;

        unsigned char               expiry_check;

        unsigned long long          expiry_time;

        unsigned char               gen_loc_in_cert_check;
```

```
            unsigned char              gen_loc_dis_check;

            double                     validity_dis;

            _3dloc                     gen_loc;

            unsigned int               crl_tolerance;

            unsigned char              cert_exp_check;

            unsigned char              signerid8[8];

            signature_t                sig;

        } sec_sec_data_verification_t;
```

And the fields are defined as following table.

Field	Description
sdee_id	Reserved, currently not used.
psid	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the header field has the value, then the value would be set by the primitive. The PSID value would be used to check whether the signer has the permission or not.
content_type	Reserved, currently not used.
signed_data_len	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the secure message is a signed message, then the value would be set by the primitive. The signed data would be used to verify the signature.
signed_data	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the secure message is a signed message, then the content would be set by the primitive. The signed data would be used to verify the signature.
external_data_hash	Reserved, currently not used.
hash_alg	Reserved, currently not used.
max_cert_chain_len	Reserved, currently not used.

data_hash_for_replay_check	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the secure message is a signed message, then the content would be set by the primitive. If the replay_check is set to 1, the content would be used to check if it is a replay of previous message.
replay_check	The value should be 0 or 1. It indicates whether to perform the replay check or not.
gen_time	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the header field has the value, then the value would be set by the primitive. The value would be used to perform generation time related checks.
gen_time_in_past_check	The value should be 0 or 1. It indicates whether to check the generation time is in past or not.
validity_period	The range of the allowable period for checking the generation time is too old or not. The unit is in microsecond.
gen_time_in_future_check	The value should be 0 or 1. It indicates whether to check the generation time is in future or not.
future_data_period	The range of the allowable period for checking the generation time is in the future or not. The unit is in microsecond.
expiry_check	The value should be 0 or 1. It indicates whether to check the expiration time or not.
expiry_time	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the header field has the value, then the value would be set by the primitive. The value would be used to perform expiry check.
gen_loc_in_cert_check	The value should be 0 or 1. It indicates whether to check if the generation location is in the permitted range of the signer or not.
gen_loc_dis_check	The value should be 0 or 1. It indicates whether to check if the generation location is too far or not.
validity_dis	The allowable distance for checking the generation location is nearby or not. The unit is in meter.
gen_loc	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the header field has the value, then the value would be set by the primitive. The value would be used to perform generation location related checks.
crl_tolerance	Reserved, currently not used.

cert_exp_check	The value should be 0 or 1. It indicates whether to check the expiration time of the signer certificate chain or not.
signerid8	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the header field has theinformation of the signer, then the value would be set by the primitive. The signerid8 would be used to indicate the signer and check related fields of the signer certificate.
sig	If the structure is one of the input parameter of sec_sec_data_preprocessing() called before and the secure message is a signed message, then the signature would be set by the primitive. The signature would be used to check the validity of the message.

The structure of the parameter encrypt_info in sec_encrypt is defined as following:

```
    typedef struct
    {
        unsigned int            data_len;
        unsigned char*          data;
        dot2_data_type_t        data_type;
        unsigned char           enc_key_type;
        unsigned int            scmh;
        unsigned int            recp_cert_num;
        unsigned char*          recp_cert_id_list;
        unsigned char*          signed_data_recp_info;
        unsigned char           resp_enc_key[8];
        dot2_ec_point_choice_t      ecpoint_format;
        unsigned int          enc_data_len;

        unsigned char*        enc_data;

        unsigned int          fail_cert_num;

        unsigned char*        fail_cert_id_list;

    } sec_encrypt_t;
```

And the fields are defined as following table:

Field	Description
data_len	The length in bytes of the input data.
data	The data to be encrypted.
data_type	If the data type is UNSECURE, then the primitive would pack the input data into an Ieee1609Dot2Data structure before encode it into an encrypted message.
enc_key_type	Reserved, currently not used.
scmh	Reserved, currently not used.
recp_cert_num	The number of recipients.
recp_cert_id_list	The array of certificate id (hashid8) of the recipients.
signed_data_recp_info	Reserved, currently not used.
resp_enc_key	Reserved, currently not used.
ecpoint_format	The value should be COMPRESSED or UNCOMPRESSED. It indicates the point format for encoding the RecipientInfos.

enc_data_len	The length in bytes of the encrypted data. NOTE: when calling the primitive, the enc_data_len should be set to the value of the maximum length of enc_data. If the encrypted data length is larger than the input value, it would fail to encrypt the message. If the encryption succeeded, the enc_data_len would be set to the actual length of the encrypted data.
enc_data	The encrypted data. NOTE: the user should allocate enough space for the security module to fill the encrypted message before calling the primitive.
fail_cert_num	The number of recipients with which failed to encrypt.
fail_cert_id_list	The array of certificate id (hashid8) of the recipients with which failed to encrypt.

The structure of the parameter decrypt_info in sec_decrypt is defined as following:

```
typedef struct

{

      unsigned int          data_len;

      unsigned char*        data;

      unsigned int          cmh;

      unsigned char         signed_data_recp_info[32];



      unsigned int          decrypted_data_len;

      unsigned char*        decrypted_data;

} sec_encrypt_t;
```

And the fields are defined as following table.

Field	Description
data_len	The length in bytes of the input data.
data	The data to be encrypted.
cmh	The primitive would use the certificate and key binded with input cmh to decrypt the message.
signed_data_recp_info	Reserved, currently not used.
decrypted_data_len	The length in bytes of the decrypted data. NOTE: when calling the primitive, the decrypted_data_len should be set to the value of the maximum length of decrypted_data. If the decrypted data length is larger than the input value, it would fail to decrypt the message. If the decryption succeeded, the decrypted_data_len would be set to the actual length of the decrypted data.
decrypted_data	The decrypted data. NOTE: the user should allocate enough space for the security module to fill the decrypted message before calling the primitive.

The protocol stack also provides two primitives to encode/decode secure messages more easily if the user does not want to setup lots of parameters. The primitives are listed below.

```
int     dot2_encode_msg(unsigned int i_psid, dot2_encode_msg_t*
        para);
int     dot2_decode_msg(unsigned int i_psid, dot2_decode_msg_t*
        para);
```

The structure of the parameter para in dot2_encode_msg is defined as following.

```
typedef struct

{

        unsigned int            i_cmh;

        unsigned char           i_sign_flag;

        unsigned int            i_msg_len;

        unsigned char*          i_msg;

        unsigned int            i_lifetime;

        unsigned int            o_msg_len;

        unsigned char*          o_msg;

} dot2_encode_msg_t;
```

And the fields are defined as following table.

Field	Description
i_cmh	The primitive would use the certificate and key binded with input cmh to encode the message.
i_sign_flag	The value indicates whether to sign the message or not.
i_msg_len	The length in bytes of the input data.
i_msg	The data to be encoded.
i_lifetime	If the input parameter psid is not BSM_PSID, then the primitive would use the value of the i_lifetime as the expiration field in the header and the expiration time would be set to the current time + the amount of i_lifetime (unit: ms).

o_msg_len	The length in bytes of the encoded data. NOTE: when calling the primitive, the o_msg_len should be set to the value of the maximum length of o_msg. If the encoded data length is larger than the input value, it would fail to encode the message. If the encoding process succeeded, the o_msg_len would be set to the actual length of the encoded data.
o_msg	The encoded data. NOTE: the user should allocate enough space for the security module to fill the encoded message before calling the primitive.

The structure of the parameter para in dot2_decode_msg is defined as following:

```
typedef struct

{

    unsigned char         i_verify_flag;

    unsigned int          i_msg_len;

    unsigned char*        i_msg;

    unsigned int          o_msg_len;

    unsigned char*        o_msg;

    unsigned long long    o_generation_time;

    unsigned long long    o_expiration_time;

    unsigned int          o_next_crl_time;

} dot2_decode_msg_t;
```

And the fields are defined as following table.

Field	Description
i_verify_flag	The value indicates whether to verify the message or not if it is a signed message.
i_msg_len	The length in bytes of the input data.
i_msg	The data to be decoded.
o_msg_len	The length in bytes of the decoded data. NOTE: when calling the primitive, the o_msg_len should be set to the value of the maximum length of o_msg. If the decoded data length is larger than the input value, it would fail to decode the message. If the decoding process succeeded, the o_msg_len would be set to the actual length of the decoded data.
o_msg	The decoded data. NOTE: the user should allocate enough space for the security module to fill the decoded message before calling the primitive.
o_generation_time	If decoded successfully, the o_generation_time would be set to the value retrieved from the secure header if the header has that field.
o_expiration_time	If decoded successfully, the o_expiration_time would be set to the value retrieved from the secure header if the header has that field.
o_next_crl_time	Reserved, currently not used.

4.3 Example code

Please refer to cmd_test_us_sec.c.

5. SAE J2735 BSM PROGRAMMING API (r63)

To support interoperability among DSRC applications, SAE J2735 standard defines standardized messages. One of them is Basic Safety Message (BSM), as per release version 63.

5.1 Basic Safety Message (BSM)

The basic safety message (BSM) encoding, using ASN.1 PER encoding, is defined below. Part I data shall be included in every BSM. Part II data items are optional for a given BSM and are included as needed according to policies that are beyond the scope of J2735 standard. A BSM without Part II optional content is a valid message.

```
typedef struct BasicSafetyMessage

{

    /* Part I, Sent at all times with each message */

    BSMcoreData_t     coreData;



    /*Part II Content*/

    struct BasicSafetyMessage__partII

    {
```

```
                A_SEQUENCE_OF(struct PartIIcontent) list;
            } *partII;
        } BasicSafetyMessage_t;
```

5.2 Data Elements of BSM

The data structure bsm_standard_item of BSM data elements is defined as follows:

```
typedef struct bsm_standard_item_str
{
    unsigned char mask_partII:1,
            mask_regional:1,
            rest:6;

    /* (1) Part I, core_data */
    bsm_core_data core_data;

    /* (2) PartII, Content, SEQUENCE (SIZE(1..8)) */
    unsigned char partII_num;
    partII_item *partII;
}bsm_standard_item;
```

```
typedef struct bsm_core_data_str

{

        unsigned char msg_cnt;

        unsigned char id[4];

        unsigned short sec_mark;

        int lat;

        int longitude;

        int elev;

        positional_accuracy_item accuracy;

        unsigned char transmission_state;

        unsigned short speed;

        unsigned short heading;

        signed char angle;

        acceleration_set_4way_item accel_set;

        brake_system_status_item brakes;

        vehicle_size_item size;

}bsm_core_data;



typedef struct positional_accuracy_str

{

        unsigned char semi_major;
```

```
    unsigned char semi_minor;
    unsigned short orientation;
}positional_accuracy_item;

typedef struct acceleration_set_4way_str
{
    short longitude;
    short lat;
    signed char vert;
    short yaw;
}acceleration_set_4way_item;

typedef struct brake_system_status_str
{
    unsigned char wheel_brakes;
    unsigned char traction;
    unsigned char abs;
    unsigned char scs;
    unsigned char brake_boost;
    unsigned char aux_brakes;
}brake_system_status_item;
```

```
typedef struct vehicle_size_str
{
      unsigned short width;
      unsigned short length;
}vehicle_size_item;

typedef struct partII_str
{
      unsigned char partII_id;
      vehicle_safety_extensions_item *vse;
      special_vehicle_extensions_item *spve;
      supplemental_vehicle_extensions_item *suve;
}partII_item;

typedef struct vehicle_safety_extensions_str
{
      unsigned char mask_events:1,
                        mask_path_history:1,
                        mask_path_prediction:1,
                        mask_lights:1,
                        rest:4;
```

```
        unsigned short events;
        path_history_item *path_history;
        path_prediction_item path_prediction;
        unsigned short lights;
}vehicle_safety_extensions_item;

typedef struct path_history_str
{
        unsigned char mask_initial_position:1,
                      mask_curr_gps_status:1,
                      reserved:6;

        full_position_vector_item *initial_position;
        unsigned char curr_gps_status;
        unsigned char crumb_data_num;
        path_history_point_item *crumb_data;
}path_history_item;

typedef struct full_position_vector_str
{
        unsigned char mask_utc_time:1,
                    mask_elevation:1,

                    mask_heading:1,

                    mask_speed:1,

                    mask_pos_accuracy:1,

                    mask_time_confidence:1,

                    mask_pos_confidence:1,

                    mask_speed_confidence:1;


        ddate_time_item *utc_time;

        long longitude;

        long lat;

        int elevation;

        unsigned short heading;

        transmission_speed_item speed;

        positional_accuracy_item pos_accuracy;

        unsigned char time_confidence;

        position_confidence_set_item pos_confidence;

        speed_heading_throttle_confidence_item speed_confidence;

}full_position_vector_item;



typedef struct ddate_time_str
```

```
{
        unsigned char mask_year:1,
                      mask_month:1,
                      mask_day:1,
                      mask_hour:1,
                      mask_minute:1,
                      mask_second:1,
                      mask_offset:1,
                      reserved:1;

        unsigned short year;
        unsigned char month;
        unsigned char day;
        unsigned char hour;
        unsigned char minute;
        unsigned short second;
        short offset;
}ddate_time_item;

typedef struct transmission_speed_str
{
        unsigned char transmission_state;

        unsigned short speed;

}transmission_speed_item;


typedef struct position_confidence_set_str

{

        unsigned char pos;

        unsigned char elevation;

}position_confidence_set_item;


typedef struct speed_heading_throttle_confidence_str

{

        unsigned char heading;

        unsigned char speed;

        unsigned char throttle;

}speed_heading_throttle_confidence_item;


typedef struct path_history_point_str

{

        unsigned char mask_speed:1,

                      mask_pos_accuracy:1,
                      mask_heading:1,
                      reserved:5;

        int lat_offset;
        int lon_offset;
        int elevation_offset;
        unsigned short time_offset;
        unsigned short speed;
        positional_accuracy_item pos_accuracy;
        unsigned char heading;
}path_history_point_item;

typedef struct path_prediction_str
{
        int radius_of_curve;
        unsigned char confidence;
}path_prediction_item;

typedef struct special_vehicle_extensions_str
{
        unsigned char mask_vehicle_alerts:1,
                            mask_description:1,
                            mask_trailers:1,
                            rest:5;

        emergency_details_item vehicle_alerts;
        event_description_item description;
        trailer_data_item trailers;
}special_vehicle_extensions_item;

typedef struct emergency_details__str
{
        unsigned char mask_events:1,
                            mask_response_type:1,
                            rest:6;

        unsigned char ssp_rights;
        unsigned char siren_use;
        unsigned char lights_use;
        unsigned char multi;
        privileged_events_item events;
        unsigned char response_type;
```

```
}emergency_details_item;

typedef struct privileged_events_str
{
     unsigned char ssp_rights;
     unsigned short event;
}privileged_events_item;

typedef struct event_description_str
{
     unsigned char mask_description:1,
                   mask_priority:1,
                   mask_heading:1,
                   mask_extent:1,
                   mask_regional:1,
                   rest:3;

     unsigned short type_event;
     unsigned char description_num;
     unsigned short description[8];
     unsigned short heading;
```

```
      unsigned char extent;
}event_description_item;

typedef struct trailer_data_str
{
      unsigned char ssp_rights;
      pivot_point_description_item connection;
      unsigned char units_num;
      trailer_unit_description_item *units;
} trailer_data_item;

typedef struct pivot_point_description_str
{
      short pivot_offset;
      unsigned short pivot_angle;
      char pivots;
} pivot_point_description_item;

typedef struct trailer_unit_description_str
{
      unsigned char mask_height:1,
                        mask_mass:1,
                        mask_bumper_heights:1,
                        mask_center_of_gravity:1,
                        mask_rear_pivot:1,
                        mask_rear_wheel_offset:1,
                        mask_elevation_offset:1,
                        mask_crumb_data:1;

      char is_dolly;
      unsigned short width;
      unsigned short length;
      unsigned char height;
      unsigned char mass;
      bumper_heights_item bumper_heights;
      unsigned char center_of_gravity;
      pivot_point_description_item front_pivot;
      pivot_point_description_item rear_pivot;
      short rear_wheel_offset;
      node_xy_24b_item position_offset;
      char elevation_offset;
      unsigned char crumb_data_num;
```

```
    trailer_history_point_item *crumb_data;
} trailer_unit_description_item;

typedef struct bumper_heights_str
{
    unsigned char front;
    unsigned char rear;
}bumper_heights_item;

typedef struct node_xy_24b_str
{
    int x;
    int y;
}node_xy_24b_item;

typedef struct trailer_history_point_str
{
    unsigned char mask_elevation_offset:1,
                      mask_heading:1,
                      rest:6;
    unsigned short pivot_angle;

    unsigned short time_offset;

    node_xy_24b_item position_offset;

    char elevation_offset;

    unsigned char heading;

} trailer_history_point_item;


typedef struct supplemental_vehicle_extensions_str

{

    unsigned char mask_classification:1,

                      mask_class_details:1,

                      mask_vehicle_data:1,

                      mask_weather_report:1,

                      mask_weather_probe:1,

                      mask_obstacle:1,

                      mask_status:1,

                      mask_speed_profile:1;

    unsigned char mask_the_rtcm:1,

                      mask_regional:1,

                      rest:6;
    unsigned char classification;
    vehicle_classification_item class_details;
    vehicle_data_item vehicle_data;
    weather_report_item weather_report;
    weather_probe_item weather_probe;
    obstacle_detection_item obstacle;
    disabled_vehicle_item status;
    unsigned char speed_profile_num;
    unsigned char speed_profile[20];
    rtcm_package_item the_rtcm;
}supplemental_vehicle_extensions_item;


typedef struct vehicle_classification_str
{
    unsigned char mask_key_type:1,
                      mask_role:1,
                      mask_iso3883:1,
                      mask_hpms_type:1,
                      mask_vehicle_type:1,
                      mask_response_equip:1,
                      mask_responder_type:1,
                      mask_fuel_type:1;
    unsigned char mask_regional:1,
                      rest:7;

    unsigned char key_type;
    unsigned char role;
    unsigned char iso3883;
    unsigned char hpms_type;
    unsigned short vehicle_type;
    unsigned short response_equip;
    unsigned short responder_type;
    unsigned char fuel_type;
} vehicle_classification_item;

typedef struct vehicle_data_str
{
    unsigned char mask_height:1,
               mask_bumpers:1,
               mask_mass:1,
               mask_trailer_weight:1,
               rest:4;
    unsigned char height;
    bumper_heights_item bumpers;
    unsigned char mass;
    unsigned short trailer_weight;
}vehicle_data_item;

typedef struct weather_report_str
{
    unsigned char mask_rain_rate :1,
               mask_precip_situation :1,
               mask_solar_radiation :1,
               mask_friction :1,
               mask_road_friction :1,
               rest:3;

    unsigned char is_raining;
    unsigned short rain_rate;
    unsigned char precip_situation;
    unsigned short solar_radiation;
    unsigned char friction;
    unsigned char road_friction;
```

```
}weather_report_item;


typedef struct weather_probe_str

{

    unsigned char mask_air_temp :1,

            mask_air_pressure :1,

            mask_rain_rates :1,

            rest:5;


    unsigned char air_temp;

    unsigned char air_pressure;

    wiper_set_item rain_rates;

} weather_probe_item;


typedef struct wiper_set_str

{

    unsigned char mask_status_rear :1,

            mask_rate_rear :1,

            rest:6;


    unsigned char status_front;
    unsigned char rate_front;
    unsigned char status_rear;
    unsigned char rate_rear;
} wiper_set_item;

typedef struct obstacle_detection_str
{
    unsigned char mask_description :1,
               mask_location_details :1,
               mask_vert_event :1,
               rest:5;

    unsigned short ob_dist;
    unsigned short ob_direct;
    unsigned short description;
    unsigned short location_details;
    ddate_time_item date_time;
    char vert_event;
} obstacle_detection_item;

typedef struct disabled_vehicle_str
```

```
{
     unsigned char mask_location_details :1,
                 rest:7;


     unsigned short status_details;

     unsigned short location_details;

} disabled_vehicle_item;


typedef struct rtcm_package_str

{

     unsigned char mask_rtcm_header:1,

                   reserved:7;


     rtcm_header_item rtcm_header;

     unsigned char msgs_num;

     rtcm_message_item *msgs;

}rtcm_package_item;


typedef struct rtcm_header_str

{

     unsigned char status;
```

```
        antenna_offset_set_item offset_set;

}rtcm_header_item;


typedef struct antenna_offset_set_str

{

        short ant_offset_x;

        short ant_offset_y;

        short ant_offset_z;

}antenna_offset_set_item;


typedef struct rtcm_message_str

{

        char rtcm_message[1023];

} rtcm_message_item;
```

bsm_standard_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
core_data	x	x	x	x	Part I, Sent at all times with each message
partII_num	1	8	x	x	Part II Content SEQUENCE (SIZE(1..8))
partII	x	x			

bsm_core_data					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
msg_cnt	0	127	x	x	The DE_MsgCount data element is used to provide a sequence number within a stream of messages with the same DSRCmsgID and from the same sender.
id[4]	x	x	x	x	This is the 4 octet random device identifier, called the TemporaryID
sec_mark	0	65535	miliseconds	65535	The DSRC second expressedinthisdata element consists of integer values from zero to 60999, representing the milliseconds within a minute.

lat	-900000000	900000001	1/10 micro degree	900000001	The geographic latitude of an object, expressed in 1/10th integer microdegrees, as a 31 bit value, and with reference to the horizontal datum then in use.
longitude	-1799999999	1800000001	1/10 micro degree	1800000001	The geographic longitude of an object, expressed in 1/10th integer microdegrees, as a 32-bit value, and with reference to the horizontal datum then in use.
elev	-4096	61439	10 cm	-4096	The DE_Elevation data element represents the geographic position above or below the reference ellipsoid.
accuracy	x	x	x	x	The DF_ PositionalAccuracy data frame consists of various parameters of quality used to model the accuracy of the positional determination with respect to each given axis.
transmission_ state	0	7	x	x	The DE_ TransmissionState data element is used to provide the current state of the vehicle transmission.

speed	0	8191	0.02 m/s	8191	This data element represents the vehicle speed expressed in unsigned units of 0.02 meters per second.
heading	0	28800	0.0125 degrees	28800	The DE_Heading data element provides the current heading of the sending device
angle	-126	127	1.5 degrees	127	The angle of the driver's steering wheel.
accel_set	x	x	x	x	This data frame is a set of acceleration values in 3 orthogonal directions of the vehicle and with yaw rotation rates, expressed as a structure.
brakes	x	x	x	x	The Brake System Status data frame conveys a variety of information about the current brake and system control activity of the vehicle.
size	x	x	x	x	The DF_VehicleSize is a data frame representing the vehicle length and vehicle width in a single data concept.

positional_accuracy_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
semi_major	0	255	0.05m	255	The DE_SemiMajorAxisAccuracy data element is used to express the radius (length) of the semi-major axis of an ellipsoid representing the accuracy which can be expected from a GNSS system in 5cm steps.
semi_minor	0	255	0.05m	255	The DE_SemiMinorAxisAccuracy data element is used to express the radius of the semi-minor axis of an ellipsoid representing the accuracy which can be expected from a GNSS system in 5cm steps.
orientation	0	65535	360/65535 deg	65535	The DE_ SemiMajorAxisOrientation data element is used to orientate the angle of the semi-major axis of an ellipsoid representing the accuracy which can be expected from a GNSS system with respect to the coordinate system.

acceleration_set_4way_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
longitude	-2000	2001	0.01 m/s^2	2001	The DE_Acceleration data element represents the signed acceleration of the vehicle along some known axis in units of 0.01 meters per second squared.

lat	-2000	2001	0.01 m/s^2	2001	The DE_Acceleration data element represents the signed acceleration of the vehicle along some known axis in units of 0.01 meters per second squared.
vert	-127	127	0.02 G	-127	A data element representing the signed vertical acceleration of the vehicle along the vertical axis.
yaw	-32767	32767	0.01 degrees	x	The DE_YawRate data element provides the Yaw Rate of the vehicle

brake_system_status_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
wheel_ brakes	0	31	x	1	The Brake Applied Status data element indicates independently for each of four wheels whether braking is currently active. The four wheels are designated Left Front, Right Front, Left Rear, and Right Rear. The indicated status of a wheel is set to 1 if brakes are active on that wheel, or to 0 if brakes are inactive on that wheel.
traction	0	0	x	0	The DE_TractionControlStatus data element reflects the status of the vehicle traction control system.
abs	0	3	x	0	The DE_AntiLockBrakeStatus data element reflects the status of the vehicle ABS.
scs	0	3	x	0	The DE_StabilityControlStatus data element reflects the current state of the stability control system.
brake_ boost	0	2	x	0	This is a data element which, when set to the “on” state, indicates emergency braking.

aux_ brakes	0	3	x	0	The DE_AuxiliaryBrakeStatus data element reflects the status of the auxiliary brakes of the vehicle.

vehicle_size_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
width	0	1023	1 cm	0	The width of the vehicle expressed in centimeters, unsigned.
length	0	4095	1 cm	0	The length of the vehicle measured from the edge of the front bumper to the edge of the rear bumper expressed in centimeters, unsigned.

partII_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
partII_id	0	2	x	x	Indicate the ID of BSMpartIIExtension, 0 for vehicleSafetyExt, 1 for specialVehicleExt and 2 for supplementalVehicleExt.
vse	x	x	x	x	The DF_VehicleSafetyExtensions data frame is used to send various additional details about the vehicle. This data frame is used for vehicle safety applications to exchange safety information such as event flag and detailed positional information.
spve	x	x	x	x	The DF_SpecialVehicleExtensions data frame is used to send various additional optional information elements in the Part II BSM used by special vehicles.
suve	x	x	x	x	The DF_SupplementalVehicleExtensions data frame is used to send various optional additional information elements in the Part II BSM.

vehicle_safety_extensions_item					
Field name	Min	Max	LSB	unavailable	Use
events	0	8191	x	x	The Vehicle Event Flags data element conveys the sender's state with regard to a set of events. For each event, the sender has the option to set the flag to 1 if the stated criteria are met, but it is not required to do so.
path_ history	x	x	x	x	The PathHistory data frame defines a geometric path reflecting time-tagged vehicle movement over some period of time and/or distance.
path_ prediction					The DF_PathPrediction data frame allows vehicles and other type of users to share their predicted path trajectory by estimating a future path of travel.
lights	0	511	x	x	The DE_ExteriorLights data element provides the status of various exterior lights (when such data is available) encoded in a bit string which can be used to relate the current vehicle settings.

path_history_item					
Field name	Min	Max	LSB	unavailable	Use
initial_ position	x	x	x	x	A complete report of the vehicle's position, speed, and heading at an instant in time. Used in the probe vehicle message (and elsewhere) as the initial position information.

curr_gps_ status	0	255	x	1	The DE_GNSSstatus data element is used to relate the current state of a GPS/GNSS rover or base system in terms of its general health, lock on satellites in view, and use of any correction information.
crumb_ data_num	1	23	x	x	The PathHistoryPointList data frame consists of a list of PathHistoryPoint entries. Note that implementations may use fewer than the maximum number of path history points allowed. SEQUENCE (SIZE(1..23))
crumb_ data	x	x			

full_position_vector_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
utc_time	x	x	x	x	The DSRC style date is a compound value consisting of finite-length sequences of integers (not characters) of the form: “yyyy, mm, dd, hh, mm, ss (sss+)”
longitude	-1799999999	1800000001	1/10 micro degree	1800000001	The geographic longitude of an object.
lat	-900000000	900000001	1/10 micro degree	900000001	The geographic latitude of an object.
elevation	-4096	61439	10 cm	-4096	The DE_Elevation data element represents the geographic position above or below the reference ellipsoid.

heading	0	28800	0.0125 degrees	28800	The DE_Heading data element provides the current heading of the sending device.
speed	x	x	x	x	The DF_TransmissionAndSpeed data frame expresses the speed of the vehicle and the state of the transmission.
pos_accuracy	x	x	x	x	The DF_PositionalAccuracy data frame consists of various parameters of quality used to model the accuracy of the positional determination with respect to each given axis.
time_confidence	0	39	x	0	The DE_TimeConfidence data element is used to provide the 95% confidence level for the currently reported value of time, taking into account the current calibration and precision of the sensor(s) used to measure and/or calculate the value.
pos_confidence	x	x	x	x	The DF_PositionConfidenceSet data frame combines multiple related bit fields into a single concept.
speed_confidence	x	x	x	x	The DF_SpeedHeadingThrottleConfidence data frame is a single data frame combining multiple related bit fields into one concept.

date_time_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
year	0	4095	years	0	The DSRC year consists of integer values from zero to 4095 representing the year according to the Gregorian calendar date system.
month	0	12	months	0	The DSRC month consists of integer values from one to 12, representing the month within a year.
day	0	31	days	0	The DSRC style day is a simple value consisting of integer values from zero to 31.
hour	0	31	hours	31	The DSRC hour consists of integer values from zero to 23 representing the hours within a day.
minute	0	60	minutes	60	The DSRC style minute is a simple value consisting of integer values from zero to 59 representing the minutes within an hour.
second	0	65535	milliseconds	65535	The DSRC second expressed in this data element consists of integer values from zero to 60999.
offset	-840	840	minutes	0	The DSRC (time zone) offset consists of a signed integer representing an hour and minute value set from -14:00 to +14:00, representing all the world's local time zones in units of minutes.

transmission_speed_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
transmission_state	0	7	x	7	The DE_TransmissionState data element is used to provide the current state of the vehicle transmission.
speed	0	8191	0.02 m/s	8191	This data element represents the velocity of an object, typically a vehicle speed or the recommended speed of travel along a roadway.

position_confidence_set_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
pos	0	15	x	0	The DE_PositionConfidence entry is used to provide the 95% confidence level for the currently reported value of entries such as the DE_Position entries, taking into account the current calibration and precision of the sensor(s) used to measure and/or calculate the value.
elevation	0	15	x	0	The DE_ElevationConfidence data element is used to provide the 95% confidence level for the currently reported value of DE_Elevation, taking into account the current calibration and precision of the sensor(s) used to measure and/or calculate the value.

speed_heading_throttle_confidence_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
heading	0	7	x	0	The DE_HeadingConfidence data element is used to provide the 95% confidence level for the currently reported value of DE_Heading, taking into account the current calibration and precision of the sensor(s) used to measure and/or calculate the value.
speed	0	7	x	0	The DE_SpeedConfidence data element is used to provide the 95% confidence level for the currently reported value of DE_Speed, taking into account the current calibration and precision of the sensor(s) used to measure and/or calculate the value.
throttle	0	3	x	0	The DE_ThrottleConfidence data element is used to provide the 95% confidence level for the currently reported value of DE_Throttle, taking into account the current calibration and precision of the sensor(s) used to measure and/or calculate the value.

path_history_point_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
lat_offset	-131072	131071	0.1 microdegrees	-131072	An 18-bit delta offset in Lat or Long direction from the last point. The offset is positive to the East and to the North directions.
lon_offset	-131072	131071	0.1 microdegrees	-131072	An 18-bit delta offset in Lat or Long direction from the last point. The offset is positive to the East and to the North directions.
elevation_ offset	-2048	2047	10 cm	-2048	A 12-bit vertical delta offset in the Z direction from the last point.
time_offset	1	65535	10 mSec	65535	The DE_TimeOffset data element is used to convey an offset in time from a known point.
speed	0	8191	0.02 m/s	8191	This data element represents the vehicle speed.
pos_ accuracy	x	x	x	x	The DF_PositionalAccuracy data frame consists of various parameters of quality used to model the accuracy of the positional determination with respect to each given axis.
heading	0	240	1.5 degrees	240	The DE_CoarseHeading data element is used to provide a coarser sense of heading than the DE_ Heading provides.

path_prediction_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
radius_of_curve	-32767	32767	10cm	x	The entry DE_RadiusOfCurvature is a data element representing an estimate of the current trajectory of the sender.
confidence	0	200	0.5 percent	x	The entry DE_Confidence is a data element representing the general confidence of another associated value.

special_vehicle_extensions_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
vehicle_alerts	x	x	x	x	The EmergencyDetails data element combines several bit level items into a structure for efficient transmission about the vehicle during a response call.
description	x	x	x	x	The EventDescription data frame provides a short summary of an event or incident.
trailers	x	x	x	x	The DF_TrailerData data frame provides a means to describe trailers pulled by a motor vehicle and/or other equipped devices.

emergency_details_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
ssp_rights	0	31	x	x	The SSP index is used to control the data elements that follow the occurrence of the index.
siren_use	0	3	x	0	A data element which is set if any sort of audible alarm is being emitted from the vehicle.

lights_use	0	7	x	0	The DE_LightbarInUse is a data element in which the named bits are set to one if any sort of additional visible lighting-alerting system is currently in use by a vehicle.
multi	0	3	x	0	DE_MultiVehicleResponse is a data element which is set if the vehicle transmitting believes that more than one vehicle (regardless of the dispatch or command and control organization of those vehicles or their agency) are currently en-route or involved in the response to the event.
events	x	x	x	x	The DF_PrivilegedEvents data frame provides a means to describe various public safety events.
response_ type	0	6	x	x	The response type and general driving behavior which this vehicle is engaged in at the time the message is being sent.

privileged_events_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
ssp_rights	0	31	x	x	The SSP index is used to control the data elements that follow the occurrence of the index.
event	0	63	x	1	The PrivilegedEventFlags data element conveys various states of the sender (typically a DSRC-equipped vehicle) and is most often used by various types of public safety vehicles in response to a service call.

event_description_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
type_event	0	65535	x	x	The complete set of ITIS codes can be found in Volume Two of the J2540 Standard.
description_ num	1	8	x	x	Up to eight ITIS code set entries to further describe the event, give advice, or any other ITIS codes. SEQUENCE (SIZE(1..8))
description[8]	x	x			
priority	x	x	x	x	A priority for the alert message, giving urgency of this message. OCTET STRING (SIZE(1))
heading	0	65535	x	x	The DE_HeadingSlice data element is used to define a set of sixteen 22.5 degree slices of a unit circle (defined as 0~360 degrees of heading) which, when a given slice is set to one, indicates that travel, or motion, or message applicability along that slice of angles is allowed.
extent	0	15	x	x	The spatial distance over which this message applies and should be presented to the driver.

trailer_data_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
ssp_rights	0	31	x	x	The SSP index is used to control the data elements that follow the occurrence of the index.
connection	x	x	x	x	The DF_PivotPointDescription data frame is used to describe the geometric relationship between a vehicle and a trailer; or a dolly and another object to which it is connected.

units_num	1	8	x	x	The DF_TrailerUnitDescription data frame provides a physical description for one trailer or a dolly element (called a unit), including details of how it connects with other elements fore and aft. SEQUENCE (SIZE(1..8))
units	x	x			

pivot_point_description_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
pivot_offset	-1024	1023	1 cm	-1024	An 11-bit delta offset in X or Y direction from some known point.
pivot_angle	0	28800	0.0125 degrees	28800	The DE_Angle data element Angle is used to describe an angular measurement in units of degrees.
pivots	0	1	x	x	The DE_PivotingAllowed data element is a flag set to true when the described connection point allows pivoting to occur.

trailer_unit_description_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
is_dolly	0	1	x	x	A DE_IsDolly data element is a flag which is set to true to indicate that the described element is a dolly type rather than a trailer type of object.
width	0	1023	1 cm	0	The width of the vehicle expressed in centimeters, unsigned.

length	0	4095	1 cm	0	The length of the vehicle measured from the edge of the front bumper to the edge of the rear bumper expressed in centimeters, unsigned.
height	0	127	5 cm	x	The height of the vehicle, measured from the ground to the highest surface, excluding any antenna(s).
mass	0	255	500 kg	0	The DE_TrailerMass data element is used to relate the current mass of a trailer.
bumper_ heights	x	x	x	x	The DF Bumper Heights data frame conveys the height of the front and rear bumper of the vehicle or object (can also be used with trailers).
center_of_ gravity	0	127	5 cm	x	The height of the vehicle, measured from the ground to the highest surface, excluding any antenna(s).
front_pivot	x	x	x	x	The DF_PivotPointDescription data frame is used to describe the geometric relationship between a vehicle and a trailer; or a dolly and another object to which it is connected.
rear_pivot	x	x	x	x	The DF_PivotPointDescription data frame is used to describe the geometric relationship between a vehicle and a trailer; or a dolly and another object to which it is connected.

rear_wheel_ offset	-2048	2047	1 cm	-2048	A 12-bit delta offset in X, Y or Z direction from some known point. For non-vehicle centric coordinate frames of reference, non-vehicle centric coordinate frames of reference, offset is positive to the East (X) and to the North (Y) directions.
position_offset	x	x	x	x	A 24-bit node type with offset values from the last point in X and Y.
elevation_ offset	-64	63	10 cm	-64	A 7-bit vertical delta offset in the Z direction from the last point.
crumb_data_ num	1	23	x	x	The DF_TrailerHistoryPoint data frame contains a single position point for a trailer, expressed relative to the vehicle's BSM positional estimate at the same point in time. SEQUENCE (SIZE(1..23))
crumb_data	x	x			

bumper_heights_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
front	0	127	0.01 meters	x	The DE_Bumper Height data element conveys the height of one of the bumpers of the vehicle or object.
rear	0	127	0.01 meters	x	The DE_Bumper Height data element conveys the height of one of the bumpers of the vehicle or object.

node_xy_24b_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
x	-2048	2047	1 cm	-2048	A 12-bit delta offset in X, Y or Z direction from some known point.
y	-2048	2047	1 cm	-2048	A 12-bit delta offset in X, Y or Z direction from some known point.

trailer_history_point_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
pivot_angle	0	28800	0.0125 degrees	28800	The DE_Angle data element Angle is used to describe an angular measurement in units of degrees.
time_offset	1	65535	10 mSec	65535	The DE_TimeOffset data element is used to convey an offset in time from a known point.
position_offset	x	x	x	x	A 24-bit node type with offset values from the last point in X and Y.
elevation_ offset	-64	63	10 cm	-64	A 7-bit vertical delta offset in the Z direction from the last point.
heading	0	240	1.5 degrees	240	The DE_CoarseHeading data element is used to provide a coarser sense of heading than the DE_Heading provides.

supplemental_vehicle_extensions_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
classification	0	255	x	0	The BasicVehicleClass data element is used to provide a common classification system to categorize DSRC- equipped devices for various cross-cutting uses.
class_details	x	x	x	x	The DF_VehicleClassification data frame is a structure with a composite set of common classification systems used in ITS and DSRC work.
vehicle_data	x	x	x	x	The DF_VehicleData data frame is used to convey additional data about the vehicle not found in the BSM Part I data frame.
weather_report	x	x	x	x	The DF_WeatherReport data frame is used to convey weather measurments made by the sending device.
weather_probe	x	x	x	x	The DF_WeatherProbe data frame provides basic data on the air temperature and barometric pressure experienced by a vehicle, as well as the current status of the wiper systems on the vehicle, including front and rear wiper systems (where equipped) to indicate coarse rainfall levels.
obstacle	x	x	x	x	The DF_ObstacleDetection data frame is used to relate basic location information about a detect obstacle or a road hazard in a vehicles path.

status	x	x	x	x	The DF_DisabledVehicle data frame provides a means for a vehicle (or other equipped device) to describe its operational status and gross location to others using a subset of the ITIS codes.
speed_profile_num	1	20	x	x	The DE_SpeedProfileMeasurement data element represents the average measured or reported speed of a series of objects traveling in the same direction over a period of time.
speed_profile[20]	x	x			
the_rtcm	x	x	x	x	The DF_RTCMPackage data frame is used to convey RTCM messages which deal with differential corrections between users from one mobile device to another.

vehicle_classification_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
key_type	0	255	x	0	The BasicVehicleClass data element is used to provide a common classification system to categorize DSRC- equipped devices for various cross-cutting uses.
role	0	22	x	x	The BasicVehicleRole data element provides a means to indicate the current role that a DSRC device is playing.
iso3883	0	100	x	x	The DE_Iso3833VehicleType data element represents the value domain provided by ISO 3833 for general vehicle types.

hpms_type	0	15	x	0	The DE_VehicleType data element is a type list (i.e., a classification list) of the vehicle in terms of overall size.
vehicle_type	9217	9251	x	x	The ITIS enumeration list commonly referred to as “Vehicle Groups Affected” is assigned the upper octet value of [36].
response_ equip	9985	10113	x	x	The ITIS enumeration list commonly refered to as “Incident Response Equipment” is assigned the upper octet value of [39].
responder_ type	9729	9742	x	x	The ITIS enumeration list commonly refered to as “Responder Group Affected” is assigned the upper octet value of [38].
fuel_type	0	15	x	0	This data element provides the type of fuel used by a vehicle.

vehicle_data_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
height	0	127	5 cm	x	The height of the vehicle, measured from the ground to the highest surface.
bumpers	x	x	x	x	The DF Bumper Heights data frame conveys the height of the front and rear bumper of the vehicle or object (can also be used with trailers).
mass	0	255	x	255	The DE_VehicleMass data element represents the estimated weight of the vehicle over a span of stepwise linear values.

trailer_weight	0	64255	x	x	A data element re-used from the SAE J1939 standard and encoded as: 2kg/bit, 0 deg offset, Range: 0 to +128,510kg.

weather_report_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
is_raining	1	3	x	x	Indicates whether or not moisture is detected by the sensor.
rain_rate	0	65535	x	x	The rainfall, or water equivalent of snow, rate in tenths of grams per square meter per second.
precip_ situation	1	15	x	2	Describes the weather situation in terms of precipitation.
solar_radiation	0	65535	x	x	The direct solar radiation integrated over the 24 hours preceding the observation in Joules, per square meter.
friction	0	101	x	x	Indicates measured coefficient of friction in percent.
road_friction	0	50	0.02 micro	0	Coefficient of Friction of an object, typically a wheel in contact with the ground.

weather_probe_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
air_temp	0	191	x	191	The DE_AmbientAirTemperature data element is used to relate the measured Ambient Air Temperature from a vehicle or other device.
air_pressure	0	255	x	0	The DE_AmbientAirPressure data element is used to relate the measured Ambient Pressure (Barometric Pressure) from a vehicle or other device.
rain_rates	x	x	x	x	The DF_WiperSet data frame provides the current status of the wiper systems on the subject vehicle.

wiper_set_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
status_front	0	6	x	0	The current status of a wiper system on the subject vehicle.
rate_front	0	127	1 minute	x	The current rate at which wiper sweeps are taking place on the subject vehicle
status_rear	0	6	x	0	The current status of a wiper system on the subject vehicle.
rate_rear	0	127	1 minute	x	The current rate at which wiper sweeps are taking place on the subject vehicle

obstacle_detection_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
ob_dist	0	32767	1 meters	x	This data element draws from the output of a forward sensing system to report the presence of an obstacle and its measured distance from the vehicle detecting and reporting the obstacle.
ob_direct	0	28800	0.0125 degrees	28800	As a companion data element to Obstacle Distance, this data element draws from the output of a forward sensing system to report the obstacle direction from the perspective of the vehicle detecting and reporting the obstacle.
description	0	65535	x	x	The complete set of ITIS codes can be found in Volume Two of the J2540 Standard.
location_ details	7936	8191	x	x	
date_time	x	x	x	x	The DSRC style date is a compound value consisting of finite-length sequences of integers (not characters) of the form: “yyyy, mm, dd, hh, mm, ss (sss+).”
vert_event	0	31	x	1	A bit string enumerating when a preset threshold for vertical acceleration is exceeded at each wheel.

disabled_vehicle_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
status_details	0	65535	x	x	The complete set of ITIS codes can be found in Volume Two of the J2540 Standard.
location_ details	7936	8191	x	x	

rtcm_package_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
rtcm_header	x	x	x	x	The DF_RTCMheader data frame is a collection of data values used to convey RTCM information between users.
msgs_num	1	5	x	x	The RTCMmessage data element contains the stream of octets of the actual RTCM message that is being sent.
msgs	x	x			

rtcm_header_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
status	0	255	x	1	The DE_GNSSstatus data element is used to relate the current state of a GPS/GNSS rover or base system in terms of its general health, lock on satellites in view, and use of any correction information.
offset_set	x	x	x	x	The DF_AntennaOffsetSet data frame is a collection of three offset values in an orthogonal coordinate system which describe how far the electrical phase center of an antenna is in each axis from a nearby known anchor point in units of 1 cm.

antenna_offset_set_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
ant_offset_x	-2048	2047	1 cm	-2048	A 12-bit delta offset in X, Y or Z direction from some known point.
ant_offset_y	-256	255	1 cm	-256	A 9-bit delta offset in X, Y or Z direction from some known point.
ant_offset_z	-512	511	1 cm	-512	A 10-bit delta offset in X, Y or Z direction from some known point.

rtcm_message_item					
Field name	**Min**	**Max**	**LSB**	**unavailable**	**Use**
rtcm_message[1023]	x	x	x	x	The RTCMmessage data element contains the stream of octets of the actual RTCM message that is being sent. OCTET STRING (SIZE(1..1023))

5.3 BSM Applications

5.3.1 Encoding

```
int j2735r63_bsm_encode(

    bsm_standard_item *encode_item,

    unsigned char encode_buf[],

    size_t input_buf_size,

    size_t *output_buf_size,

    unsigned char print_level

);
```

Parameters:

The parameters of j2735_bsm_encode function are described in the following table.

Parameters	Description
encode_item	The input BSM data elements.
encode_buf[]	If the return value is 0, encode_buf will be a BSM encoded buffer.
input_buf_size	input_buf_size is the size of the encode_buf.
print_level	PRINT_NON: print nothing PRINT_BASIC: print encoded BSM struct

Return Value:

If the return value is equal to zero, it has encoded BSM successfully.

If the return value is less than zero, it has failed in encoding BSM.

5.3.2 Decoding

```
int j2735r63_bsm_decode(

      bsm_standard_item *return_decoded_item,

      unsigned char decode_buf[],

      size_t decode_buf_size,

      unsigned char print_level

);
```

Parameters:

The parameters of j2735_bsm_decode function are described in the following table.

Parameters	Description
return_decoded_item	If the return value is 0, return_decoded_item will be the decoded bsm_standard_item.
decode_buf[]	The BSM encoded buffer
decode_buf_size	The size of the decode_buf[]
print_level	PRINT_NON: print nothing PRINT_BASIC: print decoded BSM struct

Return Value:

If the return value is equal to zero, it has decoded BSM successfully.

If the return value is less than zero, it has failed in decoding BSM.

5.4 Example code

Pls refer to cmd_j2735r63_bsm.c

Unex Technology Corporation

11F-3, No. 100, Sec. 1, Jiafeng 11th Rd., Zhubei City,
Hsinchu County 302, Taiwan, R.O.C.

Tel. 886-3-657-8188

Fax. 886-3-657-8199

www.unex.com.tw