Document Number Unex-APG-19-008
Revision 1.3.0
Authors Ian Peng

API Manual

for

ITSG5 V2Xcast SDK

Reviewers

Department	Name	Acceptance Date	Note

Modification History

Revision	Date	Originator	Comment
1.0.0	2019/5/29	Ian Peng	Creating document
1.1.0	2019/7/8	Ian Peng	Adding descriptions for V2Xcast config manager
1.2.0	2019/8/14	Howard	Adding descriptions for GBC and security
1.2.1	2019/9/3	Ian Peng	Adding GBC and security related to V2Xcast config manager and V2Xcast config
1.3.0-d1	2019/11/11	Ian Peng	Some updates for V2Xcast Config Manager
1.3.0	2019/11/18	PC Kang	Changing the definition of return values

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

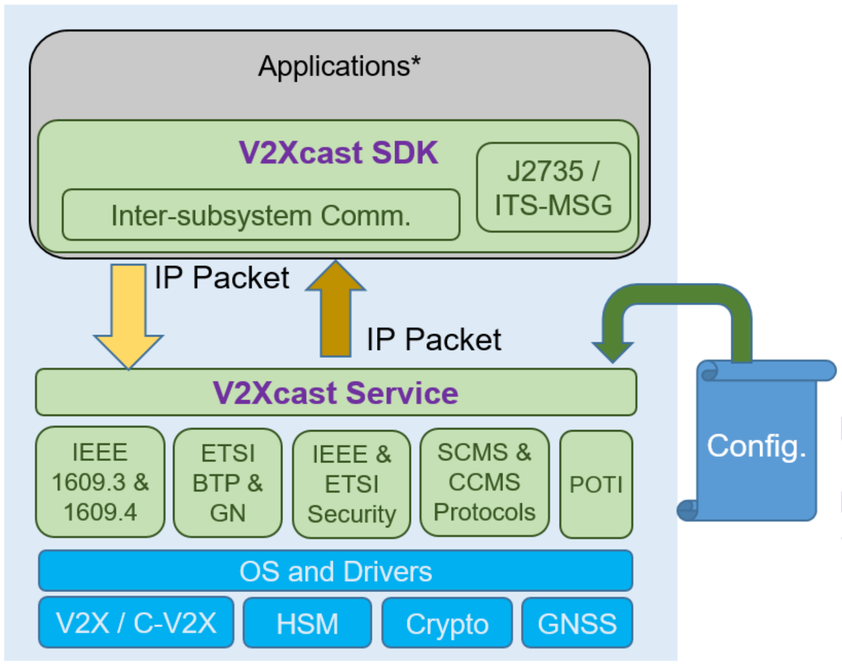

Figure 1: Architecture of V2Xcast

Users can develop V2X applications simply through V2Xcast SDK, there is a V2Xcast service on V2X devices for providing service to V2Xcast SDK. The service will handle resources of V2X networking for sending application payloads which from V2X SDK. Users will edit their own Caster config on application demand in JSON format (https://www.json.org/), so that V2Xcast Casters know the way to use resources for sending and receiving GN packets. In config file, there are 6 profile types as TABLE 1: INTRODUCTION OF V2XCAST ITSG5 profiles, except Caster profile type, rest of profile types are resource settings for Caster profiles:

Table 1: Introduction of V2Xcast ITSG5 profiles

Profile Type	Quantity	Description
Management	1(0)	Settings about GN protocol stack
Caster	16 (0~15)	A caster is for an application purpose, also, it bounds IDs of other profile types to be used
BTP	16 (0~15)	Settings for BTP usage
GN	16 (0~15)	Settings for GN usage
Channel	16 (0~15)	Settings for channel usage
Carrier	16 (0~15)	Settings for channel usage
Security	16 (0~15)	Settings for security usage
POTI	1 (0)	Settings for providing connection port to V2Xcast SDK

What is a caster? A caster is the object who has a bundle of configurations, including sending and receiving settings of GN packets, channel settings, carrier settings and POTI settings…etc. The configurations of a caster are defined in a file with JSON format. Up to 16 caster profiles can be used. By editing JSON config and invoking APIs, we can easily create a caster to handle GN transmitting and receiving. Just need to focus on the development of applications, without having to go deep into protocol-related knowledge. About the usage of APIs will be introduced in the later chapters.

V2Xcast APIs are designed to quickly set up the V2Xcast caster. Utilizing the APIs can help you to create Casters, transmit and receive packets. The features in V2Xcast SDK include:

- Creating V2Xcast caster
- Receiving messages
- Transmitting messages
- Getting POTI data
- Releasing V2Xcast caster

2. V2Xcast Config File

2.1. Format of Config File

1. The config file should follow format in JSON, users can check the format is correct or not by online JSON editor(https://jsoneditoronline.org/)
2. The first layer of the config are 6 profile types in form of JSON array, each profile type might include several setting clusters as array members
3. Elements in each array member should comply with Unex defined, please refer to PROFILE ELEMENTS AND VALUE RANGE for details
4. The value of “ID” element should be treated as array index, it should be not duplicated and should be continuous number start from zero
5. Element name is case-sensitive
6. The “Description” element can be added for note purpose, up to users

2.2. Profile Elements and Value Range

Table 2: Profile elements and value range

Profile	Element	Value Type	Value Range	Description
Management	--	Array[0]	--	1 Management Profiles at most
	GN Security Type	Character	AID/PROFILE, Unsecured	AID/PROFILE: Signing packets according to the AID profile in "gnd.json", this mode can only be set when "is_mib_its_gn_security_enabled" field in "" is setting to true Unsecured: Not signing, unsecured mode, this mode can only be set when "is_mib_its_gn_security_enabled" field in "gnd.json" is setting to false
Caster	--	Array[16]	--	16 Caster Profiles at most
	ID	Decimal	0 ~ 15	Index of the Caster setting
	Description	Character	--	Users can add note here
	BTP Profile ID	Decimal	0 ~ 15	Mapping to index of WSM Profile
	Channel Profile ID	Decimal	0 ~ 15	Mapping to index of Channel Profile
	Carrier Profile ID	Decimal	0 ~ 15	Mapping to index of Carrier Profile
	Security Profile ID	Decimal	0 ~ 15	Mapping to index of Security Profile
BTP	--	Array[16]	--	16 BTP Profiles at most
	ID	Decimal	0 ~ 15	Index of the BTP setting
	Direction	Character	TX, RX, BOTH	The setting for sending only, receiving only or bi-direction With receiving only, the following TX settings will not be used: TX Data Rate and TX Power
	BTP Type	Character	A, B	A: BTP-A B: BTP-B
	GN Profile ID	Decimal	0 ~ 16	Mapping to index of GN Profile
	Source Port	Decimal	0 ~ 65535	Port number for sending packets, using by BTP-A only
	Destination Port	Decimal	0 ~ 65535	Port number for receiving packets
	TX Power	Decimal	0 ~ 20	Transmission power (dBm)
	TX Data Rate	Decimal	6, 9, 12, 18, 24, 36, 48, 54	Transmission data rate (500kbps)
GN	--	Array[16]	--	16 GN Profiles at most
	ID	Decimal	0 ~ 15	Index of the GN setting
	Transport Type	Character	SHB, GBC	Setting for transport type
	TX Traffic Class ID	Decimal	0 ~ 3	
	TX Maximum Packet Lifetime	Decimal	0 ~ 600	(Second)
	TX Repetition Interval	Decimal	0, 100 ~ 65535	(Millisecond) disable: 0 enable: 100 ~ 65535
	TX Maximum Repetition Time	Decimal	0, 100 ~ 65535	(Millisecond) disable: 0 enable: 100 ~ 65535
	TX Maximum Hop Limit	Decimal	1 ~ 10	Used by GBC only
	Target Shape	Decimal	Circular, Rectangular, Ellipsoidal	Used by GBC only
	Distance A	Decimal	1 ~ 65535	(Meter) Used by GBC only
	Distance B	Decimal	1 ~ 65535	(Meter) Used by GBC only
	Angle	Decimal	0 ~ 359	(Degree) Used by GBC only
Channel	--	Array[16]	--	16 Channel Profiles at most
	ID	Decimal	0 ~ 15	Index of the channel setting
	Radio Interface Number	Decimal	0	0: Radio 0
	Channel Number	Decimal	180	Channel for sending or receiving
Carrier	--	Array[16]	--	16 Carrier Profiles at most
	ID	Decimal	0 ~ 15	Index of the carrier setting
	Service Port	Decimal	0 ~ 65535	Connection Port number for V2Xcast SDK

POTI	--	Array[1]	--	1 POTI Profiles at most
	ID	Decimal	0	Index of the POTI setting
	Type	Character	Publisher	source type of POTI
Security	--	Array[16]	--	16 Security Profiles at most
	ID	Decimal	0	Index of the Security setting
	AID	Decimal	0 ~ 0xFFFFFFFF	Signing packets according to the AID. Note that AID settings in the “Security Profile” should be included in AID settings of “gnd.config”, and an AID can only be executed once in the meanwhile, for AID settings in “gnd.config”, please refer to “Unex-APG-ETSI-GN-BTP.pdf” for details 0xFFFFFFFF: for message sets that security unsupported

2.3. ITSG5 Config File Example in JSON

```
{
  "Management Profile": [
    {
      "ID": 0,
      "GN Security Type": "Unsecured"
    }
  ],
  "Caster Profile": [
    {
      "ID": 0,
      "Description": "CAM TX, Channel 180, Caster service on port 7777",
      "BTP Profile ID": 0,
      "Channel Profile ID": 0,
      "Carrier Profile ID": 0,
      "Security Profile ID": 0
    },
    {
      "ID": 1,
      "Description": "CAM RX, Channel 180, Caster service on port 8888",
      "BTP Profile ID": 1,
      "Channel Profile ID": 0,
      "Carrier Profile ID": 1,
      "Security Profile ID": 0
    },
    {
      "ID": 2,
      "Description": "DENM TX, Channel 180, Caster service on port 9999",
      "BTP Profile ID": 2,
      "Channel Profile ID": 0,
      "Carrier Profile ID": 2,
      "Security Profile ID": 1
    },
    {
      "ID": 3,
      "Description": "DENM RX, Channel 180, Caster service on port 10101",
      "BTP Profile ID": 3,
      "Channel Profile ID": 0,
      "Carrier Profile ID": 3,
      "Security Profile ID": 1
    },
    {
      "ID": 4,
      "Description": "GPC(RTCMEM) TX, Channel 180, Caster service on port 
10000",
      "BTP Profile ID": 4,
      "Channel Profile ID": 0,
      "Carrier Profile ID": 4,
      "Security Profile ID": 2
    },
    {
      "ID": 5,
      "Description": "TLM(SPATEM) TX, Channel 180, Caster service on port 
10001",
      "BTP Profile ID": 5,
      "Channel Profile ID": 0,
      "Carrier Profile ID": 5,
      "Security Profile ID": 2
    }
  ],
  "BTP Profile": [
    {
      "ID": 0,
      "Description": "CAM TX service, BTP-B to port 2001, 6Mbps",
      "Direction": "TX",
      "BTP Type": "B",
      "GN Profile ID": 0,
      "Source Port": 2001,
      "Destination Port": 2001,
      "TX Power":20,
      "TX Data Rate":12
    },
    {
      "ID": 1,
      "Description": "CAM RX service, recv from port 2001",
      "Direction": "RX",
      "BTP Type": "B",
      "GN Profile ID": 1,
      "Source Port": 2001,
      "Destination Port": 2001
    },
    {
      "ID": 2,
      "Description": "DENM TX service, BTP-B to port 2002, 6Mbps",
      "Direction": "TX",
      "BTP Type": "B",
      "GN Profile ID": 2,
      "Source Port": 2002,
      "Destination Port": 2002,
      "TX Power":20,
      "TX Data Rate":12
    },
    {
      "ID": 3,
      "Description": "DENM RX service, BTP-B to port 2002, 6Mbps",
      "Direction": "RX",
      "BTP Type": "B",
      "GN Profile ID": 2,
      "Source Port": 2002,
      "Destination Port": 2002
    },
    {
      "ID": 4,
      "Description": "GPC(RTCMEM) TX service, BTP-B to port 2013, 6Mbps",
      "Direction": "TX",
      "BTP Type": "B",
      "GN Profile ID": 3,
      "Source Port": 2013,
      "Destination Port": 2013,
      "TX Power":20,
      "TX Data Rate":12
    },
    {
      "ID": 5,
      "Description": "TLM(SPATEM) service, BTP-B to/from port 2004, 6Mbps",
      "Direction": "BOTH",
      "BTP Type": "B",
      "GN Profile ID": 3,
      "Source Port": 2004,
      "Destination Port": 2004,
      "TX Power":20,
      "TX Data Rate":12
    }
  ],
  "GN Profile": [
    {
      "ID": 0,
      "Description": "SHB GN TX setting",
      "Transport Type": "SHB",
      "TX Traffic Class ID": 2,
      "TX Maximum Packet Lifetime": 1,
      "TX Repetition Interval": 0,
      "TX Maximum Repetition Time": 0
    },
    {
      "ID": 1,
      "Description": "SHB GN RX setting",
      "Transport Type": "SHB",
      "TX Traffic Class ID": 2,
      "TX Maximum Packet Lifetime": 1,
      "TX Repetition Interval": 0,
      "TX Maximum Repetition Time": 0
    },
    {
      "ID": 2,
      "Description": "DENM",
      "Transport Type": "GBC",
      "TX Traffic Class ID": 1,
      "TX Maximum Packet Lifetime": 1,
      "TX Repetition Interval": 0,
      "TX Maximum Repetition Time": 0,
      "TX Maximum Hop Limit": 1,
      "Target Shape": "Circular",
      "Distance A": 1000,
      "Distance B": 1000,
      "Angle": 0
    },
    {
      "ID": 3,
      "Description": "TLM, GPC",
      "Transport Type": "GBC",
      "TX Traffic Class ID": 3,
      "TX Maximum Packet Lifetime": 1,
      "TX Repetition Interval": 0,
      "TX Maximum Repetition Time": 0,
      "TX Maximum Hop Limit": 1,
      "Target Shape": "Circular",
      "Distance A": 400,
      "Distance B": 400,
      "Angle": 0
    }
  ],
  "POTI Profile": [
    {
      "ID": 0,
      "Description": "Internal POTI",
      "Type": "Publisher"
    }
  ],
  "Channel Profile": [
    {
      "ID": 0,
      "Description": "Channel setting",
      "Radio Interface Number": 0,
      "Channel Number": 180
    }
  ],
  "Security Profile": [
    {
      "ID": 0,
      "Description": "For CAM service",
      "AID": 36
    },
    {
      "ID": 1,
      "Description": "For DENM service",
      "AID": 37
    },
    {
      "ID": 2,
      "Description": "For Messages types that security unsupported",
      "AID": 4294967295
    }
  ],
  "Carrier Profile": [
    {
      "ID": 0,
      "Service Port": 7777
    },
    {
      "ID": 1,
      "Service Port": 8888
    },
    {
      "ID": 2,
      "Service Port": 9999
    },
    {
      "ID": 3,
      "Service Port": 10101
    },
    {
      "ID": 4,
      "Service Port": 10000
    },
    {
      "ID": 5,
      "Service Port": 10001
    }
  ]
}
```

2.4. Profile Mapping

V2Xcast SDK will manipulate V2X devices via Casters, and Casters need protocol type, channel and else settings to know the way to manipulate V2X devices. Below shows how Caster1 (ID = 1) logically maps to other settings, note that Management and POTI profile will always be invoked, so Caster doesn't need specify an ID to Management and POTI profile

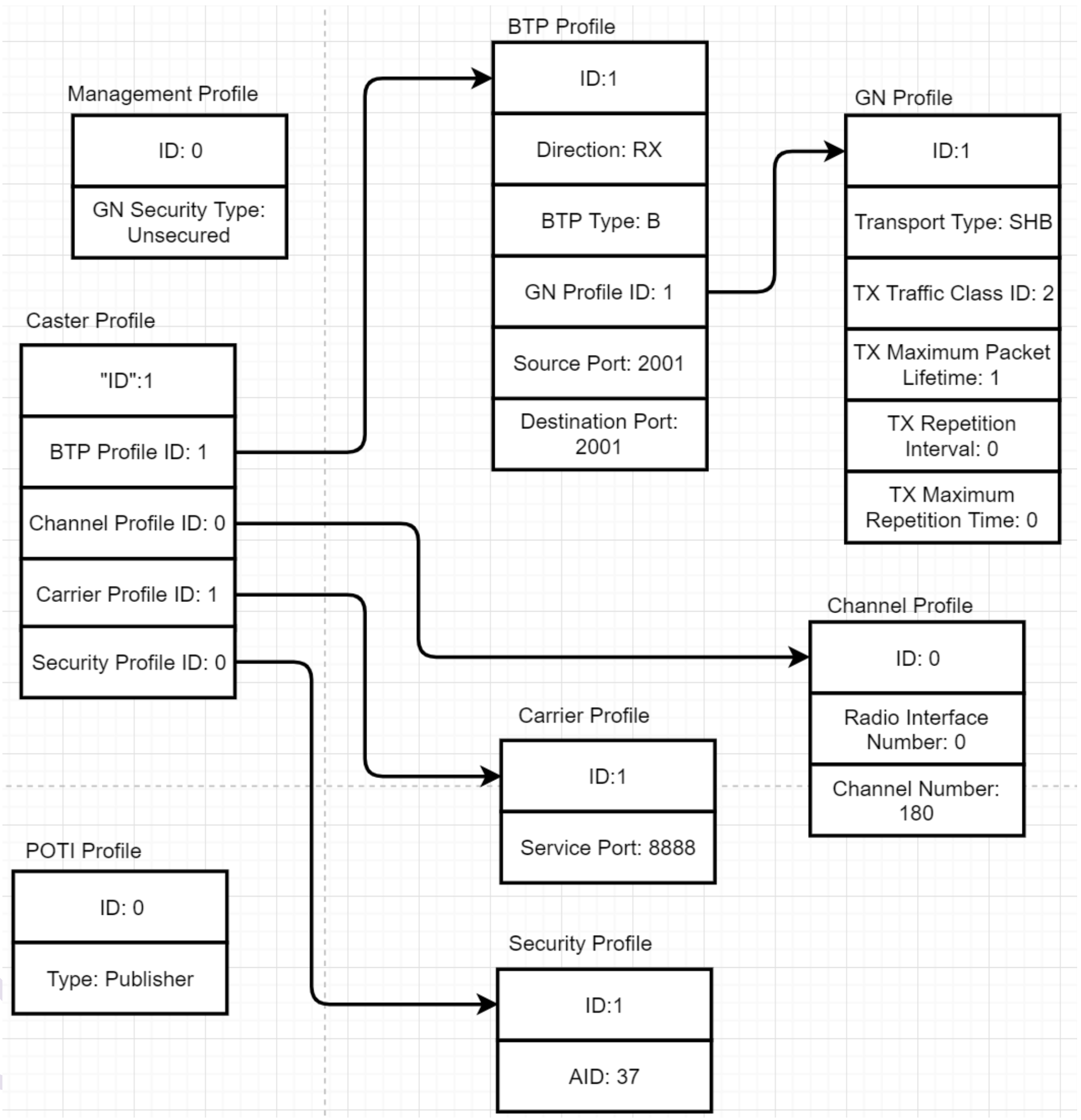

Figure 2: Example for profile mapping of Caster 1

2.5. V2Xcast Config Manager

V2Xcast Config Manager lets users to read V2Xcast config settings and system status from V2Xcast service, or to upload V2Xcast/gnd config to V2Xcast service, the application binary is in V2Xcast SDK package, i.e. "itsg5_v2xcast_sdk/bin/".

2.5.1. Reading V2Xcast Config

Command for sending read request:

```
./v2xcast_config_manager –m get
${DEVICE_IP}/${PROFILE_TYPE}/${PROFILE_ID}
```

- DEVICE_IP: IP address of the device which is running V2Xcast service
- PROFILE_TYPE: Profile types as TABLE 1: INTRODUCTION OF V2XCAST ITSG5 PROFILES
- PROFILE_ID: Index of the profile type

Table 3: Description of V2Xcast Config Manager reading request

PROFILE_TYPE	PROFILE_ID	Usage
caster	--	List all valid Caster Profile ID
caster	Valid ID	List settings of the Caster Profile ID
btp	--	List all valid BTP Profile ID
btp	Valid ID	List settings of the BTP Profile ID
gn	--	List all valid GN Profile ID
gn	Valid ID	List settings of the GN Profile ID
channel	--	List all valid Channel Profile ID
channel	Valid ID	List settings of the Channel Profile ID
carrier	--	List all valid Carrier Profile ID
carrier	Valid ID	List settings of the Carrier Profile ID
poti	--	List all valid POTI Profile ID
poti	Valid ID	List settings of the POTI Profile ID
security	--	List all valid Security Profile ID
security	Valid ID	List settings of the Security Profile ID
management	--	List all valid Management Profile ID
management	Valid ID	List settings of the Management Profile ID

For example:

```
./v2xcast_config_manager 192.168.1.3/caster
caster/0
caster/1
caster/2
caster/3
caster/4
```

```
./v2xcast_config_manager 192.168.1.3/caster/2
Caster Profile 2
  ID: 2
  PSID: 32
  Protocol Type: WSMP
  Protocol Profile ID: 2
  Channel Profile ID: 0
  Carrier Profile ID: 2
  Security Profile ID: 0
```

2.5.2. Uploading V2Xcast Config

Command for uploading V2Xcast config, remember to reboot the device for applying new config, please refer to REBOOT REQUEST for details

```
./v2xcast_config_manager –m put -f ${FILE_TO_UPLOAD}
${DEVICE_IP}/${CONFIG_TYPE}
```

- FILE_TO_UPLOAD: the uploading file with path
- DEVICE_IP: IP address of the device which is running V2Xcast service
- CONFIG_TYPE: v2xcast_config or gn_config
  - v2xcast_config will be uploaded and overwrite to path "/home/root/ext-fs/home/unex/conf/v2xcast_itsg5_config_example.json", if the config is valid
  - gn_config will be uploaded and overwrite to path "/home/root/ext-fs/home/unex/conf/gnd.json"

For example:

```
./v2xcast_config_manager –m put –f
~/config_to_upload/v2xcast_itsg5_config_example.json
192.168.1.3/v2xcast_config
Uploading ITSG5 V2Xcast config successful
```

2.5.3. Reboot Request

Command for sending reboot request:

```
./v2xcast_config_manager ${DEVICE_IP}/reboot
```

- DEVICE_IP: IP address of the device which is running V2Xcast service

For example:

```
./v2xcast_config_manager 192.168.1.3/reboot
rebooting now
```

2.5.4. Reading POTI status

Command for reading POTI status:

```
./v2xcast_config_manager ${DEVICE_IP}/poti_status
```

- DEVICE_IP: IP address of the device which is running V2Xcast service

For example:

```
./v2xcast_config_manager 192.168.1.3/poti_status
POTI status:
  Lastest Update Time: 2019-11-11 06:50:25(GMT+0)
  System Time Status: SYSTIME_SYNC_GNSS
  Fix Mode: FIX_MODE_3D
```

3. V2Xcast APIs

The V2Xcast SDK provides seven APIs are described as follows

- itsg5_caster_create
- itsg5_caster_tx
- itsg5_caster_rx
- itsg5_caster_release
- poti_caster_create
- poti_caster_release
- poti_caster_rx

3.1. APIs

3.1.1. ITSG5 Caster Create

int **itsg5_caster_create** *(caster_handler_t *p_caster_handler, caster_comm_config_t *p_config)*

Name	*itsg5_caster_create*	
Description	Creating an instance of a caster and set up the resources for the caster	
Return Value	The status of the API invoked, please refer to RETURN VALUES for details	
Parameters	**Type**	**Parameter description**
p_caster_handler	a pointer to a caster_handler_t	the handler of a caster
p_config	a pointer to a caster_comm_config_t	the communication setting, including IP and port number

Parameter Limitation	Valid Value	Description
p_caster_handler	a pointer to a caster_handler_t	the handler of a caster to be created
p_config	IP format example: "127.0.0.1" port: valid port number Note: the port number must be defined in the config file	the communication setting, including IP and port number

Data structure definition: **caster_comm_config_t**

```
typedef struct {
    uint8_t *ip;
    uint16_t port;
} caster_comm_config_t;
```

Example code: **creating ITSG5 caster**

```
/* caster handler */
caster_handler_t handler;

/* communication config */
caster_comm_config_t config = { .ip = “127.0.0.1”, .port = 7777 };

/* create ITSG5 caster */
ret = itsg5_caster_create(&handler, &config);
```

3.1.2. ITSG5 Caster TX

int **itsg5_caster_tx**(*caster_handler_t caster_handler, itsg5_tx_info_t *p_tx_info, uint8_t *buf, size_t len*)

Name	*itsg5_caster_tx*	
Description	Transmitting an ITSG5 message	
Return Value	The status of the API invoked, please refer to Return Values for details	
Parameters	**Type**	**Parameter description**
caster_handler	caster_handler_t	a created caster handler
p_tx_info	a pointer to a tx_info_t	replace the default configurations with this tx_info
buf	a pointer to an uint8_t	the message to be sent
len	size_t	number of bytes to be sent

Parameter Limitation	**Valid Value**	**Description**
caster_handler	initialized caster handler	handler context for transferring the message
p_tx_info	NULL or a pointer to an itsg5_tx_info_t	if NULL, using TX setting from JSON config, otherwise using TX setting with this tx_info parameters
buf	a pointer to the message	the message to be sent
len	message size	number of bytes to be sent

Note: the valid value of items of tx_info is described in Profile Elements and Value Range

Data structure definition: **itsg5_tx_info_t**

```
typedef struct itsg5_tx_info {
    uint16_t dest_port; /* this field is used if dest_port_is_present is true*/
    uint8_t data_rate; /* this field is used if data_rate_is_present is true*/
    int8_t tx_power; /* this field is used if tx_power_is_present is true*/
    uint8_t traffic_class_id; /* this field is used if traffic_class_id_is_present is true*/
    itsg5_tx_security_t security; /* this field is used if security_is_present is true*/
    itsg5_position_info_t position_info; /* this field is used if position_info_is_present
is true*/
    itsg5_area_typr_info_t area_type_info; /* this field is used if
area_type_info_is_present is true */
    bool dest_port_is_present;
    bool data_rate_is_present;
    bool tx_power_is_present;
    bool traffic_class_id_is_present;
    bool security_is_present;
    bool position_info_is_present;
    bool area_type_info_is_present;
} itsg5_tx_info_t;
```

Example code: **ITSG5 caster TX**

```
/* init tx_info to zero */
itsg5_tx_info_t tx_info = {0}; /* must initialize to zero */

/* To set the data rate of tx_info to 12 */
/* Another fields are depending on default value*/
tx_info.data_rate_is_present = true;
tx_info.data_rate = 12;
ret = itsg5_caster_tx (handler, &tx_info, buf, sizeof(buf));
/* using default TX setting, passing NULL as the first parameter to the function.
ret = itsg5_caster_tx (handler, NULL, buf, sizeof(buf));
*/
```

Note: It is import to initialize tx_info to zero or the behavior might be unexpected.

Note: More details please to Caster TX

3.1.3. ITSG5 Caster RX

int **itsg5_caster_rx** *(caster_handler_t caster_handler, itsg5_rx_info_t *p_rx_info, uint8_t *buf, size_t *p_len)*

Name	*itsg5_caster_rx*	
Description	receive a ITSG5 message	
Return Value	The status of the API invoked, Please refer to RETURN VALUES for details	
Parameters	**Type**	**Parameter description**
caster_handler	caster_handler_t	a created caster handler
p_rx_info	a pointer to a rx_info_t	used to receive the information of rx_info
buf	a pointer to a uint8_t	used to receive a message
p_len	a pointer to a size_t	the number of bytes received

Parameter Limitation	**Valid Value**	**Description**
caster_handler	initialized caster handler	handler context for receiving the message
p_rx_info	a pointer to a itsg5_tx_info_t	used to receive the information of rx_info
buf	a pointer to the message	the message to be sent
p_len	a pointer to a size_t	the number of bytes received

Data structure definition: **itsg5_rx_info_t**

```
typedef struct {
    struct timeval timestamp;  /* the received time of the packet from lower layer */
    uint8_t rssi; /* the receving RSSI of the packet */
    uint8_t data_rate;  /* 500kbit per sec */
    uint8_t traffic_class_id;
    uint8_t remain_hop_limit;
    itsg5_rx_security_t security;
    itsg5_position_info_t position_info;
    itsg5_area_type_info_t area_type_info;
} itsg5_rx_info_t;
```

Example code: **ITSG5 caster RX**

```
uint8_t rx_buf[RX_MAX];
itsg5_rx_info_t rx_info = {0};

ret = itsg5_caster_rx(handler, &rx_info, rx_buf, &len);
```

3.1.4. ITSG5 Caster Release

int **itsg5_caster_release***(caster_handler_t caster_handler)*

Name	*Itsg5_caster_release*	
Description	release a caster instance and clean up the resources needed for the caster instance	
Return Value	The status of the API invoked, Please refer to RETURN VALUES for details	
Parameters	**Type**	**Parameter description**
caster_handler	caster_handler_t	the caster handler to be released

Parameter Limitation	**Valid Value**	**Description**
caster_handler	initialized caster handler	the caster handler to be released

Example code: **ITSG5 caster release**

```
caster_handler_t handler;

/* initialize a caster hander, invoke itsg5_caster_create */
...
/* release the caster */
ret = itsg5_caster_release(handler);
```

3.1.5. POTI Caster Create

int **poti_caster_create** *(poti_handler_t *p_poti_handler, poti_comm_config_t *p_config)*

Name	*poti_caster_create*	
Description	Create an instance of a POTI caster and set up the resources needed for the caster	
Return Value	The status of the API invoked, please refer to RETURN VALUES for details	
Parameters	**Type**	**Parameter description**
p_poti_handler	a pointer to a poti_handler_t	the handler of a poti caster

Parameter Limitation	**Valid Value**	**Description**
p_poti_handler	a pointer to a poti_handler_t	the handler of a poti caster to be created
p_config	a pointer to a poti_comm_config_t IP format example: "127.0.0.1"	the IP of communication setting port is fixed to 55555

Data structure definition: **poti_comm_config_t**

```
typedef struct {
    uint8_t *ip;
} poti_comm_config_t;
```

Example code: **creating POTI caster**

```
/* POTI handler */
poti_handler_t handler;
poti_comm_config_t comm_config = {.ip = “127.0.0.1”};

/* create POTI caster */
ret = poti_caster_create(&handler, &comm_config);
```

3.1.6. POTI Caster RX

int **poti_caster_rx** *(poti_handler_t poti_handler, poti_fix_data_t *p_fix_data)*

Name	*poti_caster_rx*	
Description	receive a POTI fix data	
Return Value	The status of the API invoked, Please refer to RETURN VALUES for details	
Parameters	**Type**	**Parameter description**
poti_handler	poti_handler_t	a created poti caster handler
p_fix_data	a pointer to a poti_fix_data_t	used to receive the information of a POTI fix data

Parameter Limitation	**Valid Value**	**Description**
poti_handler	initialized caster handler	handler context for receiving POTI fix data
p_fix_data	a pointer to a poti_fix_data which cannot be NULL	used to receive the information of a POTI fix data

Data structure definition: **poti_fix_data_t**

```
typedef struct poti_fix_data_t {
    poti_fix_status_t status;
    poti_fix_mode_t mode;
    poti_fix_time_t time;
    /* position */
    double latitude;  /* degree, [-90.0, 90.0] */
    double longitude; /* degree, [-180.0, 180.0] */
    double altitude;  /* meter*/

    /* heading, speed */
    double course_over_ground;  /* relative to true north (clockwise) in degree. [0.0,
360.0] */
    double horizontal_speed;  /* meter per second */
    double vertical_speed;  /* meter per second */

    /* Error */
    double err_time;  /* second */
    /* horizontal position standard deviation ellipse */
    double err_smajor_axis;  /* semi-major axis length in meter */
    double err_sminor_axis;  /* semi-mijor axis length in meter */
    double err_smajor_orientation;  /* major axis direction relative to true norther
(clockwise) in degree. [0.0, 360.0] */
    double err_altitude;  /* in meter */
    double err_course_over_ground;  /* degree */
    double err_horizontal_speed;  /* meter per second */
    double err_vertical_speed;  /* meter per second */

    /* skyview */
    poti_service_tm_t skyview_time;
    int num_satellites_used;
    int num_satellites_visible[SAT_TYPE_NUM];
} poti_fix_data_t;
```

Example code: **POTI caster RX**

```
poti_fix_data_t fix_data = {0};

ret = poti_caster_rx(handler, &fix_data);
```

3.1.7. POTI Caster Release

int **poti_caster_release***(poti_handler_t poti_handler)*

Name	*poti_caster_release*	
Description	release a POTI caster instance and clean up the resources needed for the caster instance	
Return Value	The status of the API invoked, Please refer to RETURN VALUES for details	
Parameters	**Type**	**Parameter description**
poti_handler	poti_handler_t	the POTI caster handler to be released

Parameter Limitation	**Valid Value**	**Description**
poti_handler	initialized POTI handler	the POTI caster handler to be released

Example code: **POTI caster release**

```
poti_handler_t handler;

/* initialize a POTI caster handler, invoke poti_caster_create */
...
/* release the caster */
ret = poti_caster_release(handler);
```

3.2. Return Values

The definitions of return values are dynamically created by Unex. Users can find the real value by checking the file “error_code_enum.h”. The following error codes only define the errors of V2Vcast itself. The return values of V2Xcast APIs may include other values defined by different modules.

Enum	Description
ERROR_V2XCAST_SUCCESS	Success
ERROR_V2XCAST_SERVICE_COMM_CONFIG_INVAILD	using unconfigured communication setting
ERROR_V2XCAST_SERVICE_CASTER_LIMIT	caster number is up to 16
ERROR_V2XCAST_SERVICE_CASTER_EXISTS	caster service already exists
ERROR_V2XCAST_SERVICE_CASTER_CREATE_FAIL	creating caster failed
ERROR_V2XCAST_SERVICE_CASTER_RELEASE_FAIL	releasing caster failed
ERROR_V2XCAST_SERVICE_CASTER_DESTROYED	re-releasing caster service or the caster that has not been initialized yet
ERROR_V2XCAST_SERVICE_CASTER_CARRIER_FAIL	caster carrier decode/encode failed
ERROR_V2XCAST_SERVICE_CASTER_UNKNOWN_REQUEST_TYPE	getting unknown request type
ERROR_V2XCAST_SERVICE_POTI_CREATE_FAIL	creating POTI service failed
ERROR_V2XCAST_SERVICE_POTI_RX_FAIL	receiving POTI data failed
ERROR_V2XCAST_SERVICE_POTI_RELEASE_FAIL	releasing POTI service failed
ERROR_V2XCAST_SERVICE_POTI_CARRIER_FAIL	POTI carrier decode/encode failed
ERROR_V2XCAST_SESSION_CREATE_FAIL	creating session failed
ERROR_V2XCAST_SESSION_CONNECT_FAIL	unable to connect to V2X device
ERROR_V2XCAST_SESSION_CLOSE_FAIL	unable to close the connection between V2X device and host device
ERROR_V2XCAST_SESSION_TRANSMIT_FAIL	receiving/sending data from/to socket failed
ERROR_V2XCAST_SESSION_INIT_RESOURCE_FAIL	initializing GN resource failed
ERROR_V2XCAST_SESSION_RECEIVE_FAIL	receiving data from GN failed
ERROR_V2XCAST_CONFIG_GBC_BUT_NO_POSITION_INFO	lacking of position information in the tx_info
ERROR_V2XCAST_CONFIG_GBC_AREA_DISTANCE_IS_ZERO	area distance is 0
ERROR_V2XCAST_CONFIG_SECURITY_PROFILE_NOT_FOUND	V2Xcast configuration mismatch with gnd configuration
ERROR_V2XCAST_CONFIG_SSP_INVAILD_PERMISSIONS	the message over permission
ERROR_V2XCAST_CONFIG_SECURITY_FLAG_IS_ON_BUT_GN_CONFIG_SECURITY_NOT_ACTIVATED	V2Xcast security mode on but gn security off
ERROR_V2XCAST_CONFIG_SECURITY_FLAG_IS_OFF_BUT_GN_CONFI_SECURITY_ACTIVATED	V2Xcast security mode off but gn security on
ERROR_V2XCAST_CONFIG_AID_CERTIFICATE_NOT_FOUND	the corresponding certificate doesn't exist
ERROR_V2XCAST_CONFIG_AID_ALREADY_USED	the AID is used by other service
ERROR_V2XCAST_CONFIG_AID_INVALID	invalid AID
ERROR_V2XCAST_UNKNOWN	unexpected error

The best way to know the meaning of the return value is to use the function “ERROR_MSG()” to translate the return value to a readable string. The follow is an example.

Example code: **POTI caster release**

```
caster_handler_t handler;

/* initialize a DSRC caster handler, invoke dsrc_caster_create */
...
/* Sending a WSM packet */
ret = itsg5_caster_tx (handler, &tx_info, buf, sizeof(buf));
if (IS_SUCCESS(ret)) {
    printf("Success\n");
}
else {
    printf("Failed to transmit data, err code is:%d, msg = %s\n", ret, ERROR_MSG(ret));
}
```

4. Typical Using Scenarios and Examples

The typical scenarios of API usages are listed below.

4.1. Creating an ITSG5 Caster

The mainly concept is doing the creating by the itsg5_caster_create() function. Only need to set communicate config, including hostname and port. And then invoke the itsg5_caster_create() function to create caster service. The creating procedure is finished if the return value is V2XCAST_SUCCESS (value is 0). The typical scenario of creating pseudo code is shown as below.

Example code: **creating ITSG5 caster**

```
/* caster handler */
caster_handler_t handler;

/* communication config */
/* the port number must be defined in the JSON config */
comm_config_t config = { .ip = “127.0.0.1”, .port = 7777 };

/* create ITSG5 caster */
ret = itsg5_caster_create(&handler, &config);

/* create success */
if (ret == V2XCAST_SUCCESS) {
   /* do something */
}
```

4.2. Caster TX

Constructing a message which you want, and then invoking itsg5_caster_tx function to transmit the message. The tx_info parameter of itsg5_caster_tx function is optional, we can set tx_info with a specific setting or the default tx_info (defined in JSON configuration). The following examples will demonstrate various scenarios.

4.2.1. Typical Usage

4.2.1.1. Using Default tx_info Setting

1 Passing NULL as the tx_info parameter to the itsg5_caster_tx function

Note: If packet type is GBC, this feature will cause error

Example code: **default tx_info of ITSG5 caster TX**

```
uint8_t *tx_buf = NULL;
int tx_buf_len = 0;
int ret;
poti_fix_data_t fix_data = {0};

/* encode CAM as payload, please refer to project example code */
cam_encode(&tx_buf, &tx_buf_len, &fix_data);

/* encode success */
if (tx_buf != NULL) {
    /* using default TX setting, passing NULL as the tx_info parameter to the function */
    ret = itsg5_caster_tx(handler, NULL, tx_buf, (size_t)tx_buf_len);

    if (IS_SUCCESS(ret)) {
     /* do something */
    }

    /* free cam */
}
```

4.2.1.2. Using a Specific tx_info Setting

1 Initializing an itsg5_tx_info_t structure, all members of structure must be set to 0

2 Set xx_is_present to true and set xx value, xx indicates the tx_info field

3 Unset fields will be set to default values

4 Passing the pointer of tx_info as the tx_info parameter to the itsg5_caster_tx function

Example code: **tx_info of ITSG5 caster TX**

```
uint8_t *tx_buf = NULL;
int tx_buf_len = 0;
int ret;
poti_fix_data_t fix_data = {0};
itsg5_tx_info_t tx_info = {0}; /* According to C99, all tx_info members will be set to 0 */

/* encode CAM as payload, please refer to project example code */
cam_encode(&tx_buf, &tx_buf_len, &fix_data);
```

```
/* encode success */
if (tx_buf != NULL) {
    /* data rate will set to 12, others will be set to default values (JSON config) */
    tx_info.data_rate_is_present = true;
    tx_info.data_rate = 12;
    ret = itsg5_caster_tx (handler, &tx_info, tx_buf, (size_t)tx_buf_len);

    if (IS_SUCCESS(ret)) {
     /* do something */
    }

    /* free cam */
}
```

4.2.2. Security Mode

1. Initializing an itsg5_tx_info_t structure, all members of structure must be set to 0
2. Set “security_is_present” to true
3. Set SSP (Service Specific Permissions) and SSP length. The SSP permissions are defined for each message in the corresponding clause. For each octet, the most significant bit (MSB) shall be the leftmost bit
4. Invoking itsg5_caster_tx function to send secured messages

```
uint8_t *tx_buf = NULL;
int tx_buf_len = 0;
int ret;
poti_fix_data_t fix_data = {0};
itsg5_tx_info_t tx_info = {0}; /* According to C99, all tx_info members will be set to 0 */

/* encode CAM as payload, please refer to project example code */
cam_encode(&tx_buf, &tx_buf_len, &fix_data);

/* encode success */
if (tx_buf != NULL) {
    /* set security setting */
    tx_info.security_is_present = true;
    tx_info.security.ssp_len = 3;
    /*  */
    tx_info.security.ssp[0] = 0x00;
    tx_info.security.ssp[1] = 0x02;
    tx_info.security.ssp[2] = 0x60;

    ret = itsg5_caster_tx (handler, &tx_info, tx_buf, (size_t)tx_buf_len);

    if (IS_SUCCESS(ret)) {
     /* do something */
    }

    /* free cam */
}
```

4.2.3. Send GBC Packet

1. Initializing an itsg5_tx_info_t structure (tx_info), all members of structure must be set to 0
2. Set “position_info_is_present” to true

3. Set latitude and longitude, position must be set when sending GBC packet

```
uint8_t *tx_buf = NULL;
int tx_buf_len = 0;
int ret;
poti_fix_data_t fix_data = {0};
itsg5_tx_info_t tx_info = {0};  /* According to C99, all tx_info members will be set to 0 */


/* encode CAM as payload, please refer to project example code */
denm_encode(&tx_buf, &tx_buf_len, &fix_data);

/* encode success*/
if (tx_buf != NULL) {
    /* set position according to actual situation */
    tx_info.position_info_is_present = true;
    tx_info.position_info.latitude = 100;
    tx_info.position_info.longitude = 100;

    ret = itsg5_caster_tx (handler, &tx_info, tx_buf, (size_t)tx_buf_len);

    if (IS_SUCCESS(ret)) {
     /* do something */
    }

    /* free cam */
}
```

4.3. Caster RX

Receiving an ITSG5 message and rx_info. The rx_info provides more information, such as rssi, data rate and timestamp. The scenario of receiving the message is as the following pseudo code

Example code: **ITSG5 caster RX**

```
uint8_t *data;
size_t len;
itsg5_rx_info_t rx_info;
struct tm *timeinfo;
char buffer[80];
time_t t;

ret = itsg5_caster_rx(handler, &rx_info, data, &len);
/* success */
if (ret == V2XCAST_SUCCESS) {
    if (rx_info != NULL) {
        t = rx_info.timestamp.tv_sec;
        timeinfo = localtime(&t);
        strftime(buffer, 80, "%Y%m%d%H%M%S", timeinfo);
        printf("timestamp:%s\n", buffer);
        printf("rssi:%hd\n", rx_info.rssi);
        printf("data rate:%hu\n", rx_info.data_rate);
        printf("remain hop:%hu\n", rx_info.remain_hop_limit);
    }
    /* processing RX data */
      ...
}
```

4.4. Releasing ITSG5 Caster

The caster should be released if there is no need to exist. By invoking itsg5_caster_release() function, you can clean up the caster resource simply. The typical scenario of releasing pseudo code is shown as below

Example code: **ITSG5 caster release**

```
Caster_handler_t handler;

/* initialize a caster hander, invoke itsg5_caster_create */
...
/* release the caster */
ret = itsg5_caster_release(handler);
```

4.5. Get POTI Fix Data

Before getting POTI fix data, the poti_caster_create() function should be invoked for getting a handler, then invoking the poti_caster_rx() function with the handler to fill fix_data. If getting fix_data was not needed, please invoke poti_caster_release() to clean up resources of the POTI caster

Example code: **Get POTI fix data**

```
/* POTI handler */
poti_handler_t handler;
poti_comm_config_t config = {.ip = “127.0.0.1”};
poti_fix_data_t fix_data = {0};


/* create POTI caster */
ret = poti_caster_create(&handler, &config);

ret = poti_caster_rx(handler, &fix_data);
if (ret == V2XCAST_SUCCESS) {
    /* do something */
}

/* release the caster */
ret = poti_caster_release(handler);
```

5. Trouble Shooting

Q1:V2XCAST_CONFIG_SECURITY_FLAG_IS_ON_BUT_GN_CONFIG_SECURITY_NOT_ACTIVATED or V2XCAST_CONFIG_SECURITY_FLAG_IS_OFF_BUT_GN_CONFI_SECURITY_ACTIVATED?

A1: The security settings between GN config and V2Xcast config are not matched, please refer to descriptions of "GN Security Type" in 2.2 for details

Q2: V2XCAST_CONFIG_GBC_BUT_NO_POSITION_INFO?

A2: Please set position to the tx_info

Q3: V2XCAST_CONFIG_GBC_AREA_DISTANCE_IS_ZERO?

A3: Please confirm the GBC area distance of V2Xcast configuration, the distance cannot set to 0

Q4: V2XCAST_CONFIG_SECURITY_PROFILE_NOT_FOUND

A4: Only support SIGN

Q5: V2XCAST_CONFIG_AID_INVALID?

A5: AID cannot set to 0xFFFFFFFF