BG95-QuecOpen
Basic QAPI Application Note

LPWA Module Series

Rev. BG95-QuecOpen_Basic_QAPI_Application_Note_V1.0

Date: 2019-10-22

Status: Preliminary

Our aim is to provide customers with timely and comprehensive service. For any assistance, please contact our company headquarters:

Quectel Wireless Solutions Co., Ltd.
Building 5, Shanghai Business Park Phase III (Area B), No.1016 Tianlin Road, Minhang District, Shanghai, China 200233
Tel: +86 21 5108 6236
Email: info@quectel.com

Or our local office. For more information, please visit:
http://www.quectel.com/support/sales.htm

For technical support, or to report documentation errors, please visit:
http://www.quectel.com/support/technical.htm
Or email to: support@quectel.com

GENERAL NOTES

QUECTEL OFFERS THE INFORMATION AS A SERVICE TO ITS CUSTOMERS. THE INFORMATION PROVIDED IS BASED UPON CUSTOMERS' REQUIREMENTS. QUECTEL MAKES EVERY EFFORT TO ENSURE THE QUALITY OF THE INFORMATION IT MAKES AVAILABLE. QUECTEL DOES NOT MAKE ANY WARRANTY AS TO THE INFORMATION CONTAINED HEREIN, AND DOES NOT ACCEPT ANY LIABILITY FOR ANY INJURY, LOSS OR DAMAGE OF ANY KIND INCURRED BY USE OF OR RELIANCE UPON THE INFORMATION. ALL INFORMATION SUPPLIED HEREIN IS SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.

COPYRIGHT

About the Document

History

Revision	Date	Author	Description
1.0	2019-10-22	Justice HAN/Mac ZHU/ Alfred LI/Egbert XU	Initial

Contents

Table Index

1 Introduction

This document is the reference specification for the Qualcomm Application Programming Interface (QAPI) for the ThreadX OS. Working with module through QAPIs rather than AT commands allows customers to design their own QuecOpen applications more flexibly and efficiently.

This document provides the public interfaces necessary to use the features provided by the QAPIs. A functional overview and information on leveraging the interface functionality are also provided.

NOTE

In this document, BG95 refers to the corresponding module series models, including BG95M1, BG95M2 and BG95M3 models.

2 DSS Net Control APIs

This chapter provides the following APIs for DSS netctrl to interact with the underlying data control plane:

```
qapi_DSS_Init
qapi_DSS_Release
qapi_DSS_Get_Data_Srvc_Hndl
qapi_DSS_Rel_Data_Srvc_Hndl
qapi_DSS_Set_Data_Call_Param
qapi_DSS_Start_Data_Call
qapi_DSS_Stop_Data_Call
qapi_DSS_Get_Pkt_Stats
qapi_DSS_Reset_Pkt_Stats
qapi_DSS_Get_Call_End_Reason
qapi_DSS_Get_Call_Tech
qapi_DSS_Get_Current_Data_Bearer_Tech
qapi_DSS_Get_Device_Name
qapi_DSS_Get_Qmi_Port_Name
qapi_DSS_Get_IP_Addr_Count
qapi_DSS_Get_IP_Addr
qapi_DSS_Get_IP_Addr_Per_Family
qapi_DSS_Get_Link_Mtu
```

2.1. Data Types

2.1.1. Enumeration Type

2.1.1.1. enum qapi_DSS_Auth_Pref_t

Authentication preference for a PDP connection.

```
typedef enum qapi_DSS_Auth_Pref_e
{
   QAPI_DSS_AUTH_PREF_PAP_CHAP_NOT_ALLOWED_E = 0x00,
   QAPI_DSS_AUTH_PREF_PAP_ONLY_ALLOWED_E,
   QAPI_DSS_AUTH_PREF_CHAP_ONLY_ALLOWED_E,
```

```
    QAPI_DSS_AUTH_PREF_PAP_CHAP_BOTH_ALLOWED_E
} qapi_DSS_Auth_Pref_t;
```

- **Parameters**

Parameter	Description
QAPI_DSS_AUTH_PREF_PAP_CHAP_NOT_ALLOWED_E	Neither of the authentication protocols (PAP,CHAP) are allowed.
QAPI_DSS_AUTH_PREF_PAP_ONLY_ALLOWED_E	Only PAP authentication protocol is allowed.
QAPI_DSS_AUTH_PREF_CHAP_ONLY_ALLOWED_E	Only CHAP authentication protocol is allowed.
QAPI_DSS_AUTH_PREF_PAP_CHAP_BOTH_ALLOWED_E	Both PAP and CHAP authentication protocols are allowed.

2.1.1.2. enum qapi_DSS_CE_Reason_Type_t

Call end reason type. This enumeration is used in *qapi_DSS_CE_Reason_t* structure.

```
typedef enum qapi_DSS_CE_Reason_Type_e
{
QAPI_DSS_CE_TYPE_UNINIT_E = -2,
QAPI_DSS_CE_TYPE_INVALID_E = 0XFF,
QAPI_DSS_CE_TYPE_MOBILE_IP_E = 0x01,
QAPI_DSS_CE_TYPE_INTERNAL_E = 0x02,
QAPI_DSS_CE_TYPE_CALL_MANAGER_DEFINED_E = 0x03,
QAPI_DSS_CE_TYPE_3GPP_SPEC_DEFINED_E = 0x06,
QAPI_DSS_CE_TYPE_PPP_E = 0x07,
QAPI_DSS_CE_TYPE_EHRPD_E = 0x08,
QAPI_DSS_CE_TYPE_IPV6_E = 0x09
} qapi_DSS_CE_Reason_Type_t;
```

- **Parameters**

Parameter	Description
QAPI_DSS_CE_TYPE_UNINIT_E	No specific call end reason was received from the modem.
QAPI_DSS_CE_TYPE_INVALID_E	No valid call end reason was received.
QAPI_DSS_CE_TYPE_MOBILE_IP_E	Mobile IP error.

QAPI_DSS_CE_TYPE_INTERNAL_E	Data services internal error was sent by the modem.
QAPI_DSS_CE_TYPE_CALL_MANAGER_DEFINED_E	Modem Protocol internal error.
QAPI_DSS_CE_TYPE_3GPP_SPEC_DEFINED_E	3GPP specification defined error.
QAPI_DSS_CE_TYPE_PPP_E	Error during PPP negotiation.
QAPI_DSS_CE_TYPE_EHRPD_E	Error during EHRPD.
QAPI_DSS_CE_TYPE_IPV6_E	Error during IPv6 configuration.

2.1.1.3. enum qapi_DSS_Call_Param_Identifier_t

Call parameter identifier. This enumeration is used in *qapi_DSS_Set_Data_Call_Param()* function.

```
typedef enum qapi_DSS_Call_Info_Enum_e
{
   QAPI_DSS_CALL_INFO_MIN_E = 0x00,
   QAPI_DSS_CALL_INFO_UMTS_PROFILE_IDX_E,
   QAPI_DSS_CALL_INFO_APN_NAME_E,
   QAPI_DSS_CALL_INFO_USERNAME_E,
   QAPI_DSS_CALL_INFO_PASSWORD_E,
   QAPI_DSS_CALL_INFO_AUTH_PREF_E,
   QAPI_DSS_CALL_INFO_CDMA_PROFILE_IDX_E,
   QAPI_DSS_CALL_INFO_TECH_PREF_E,
   QAPI_DSS_CALL_INFO_IP_VERSION_E,
   QAPI_DSS_CALL_INFO_EXT_TECH_E,
   QAPI_DSS_CALL_INFO_MO_EXCEPTION_DATA_E,
   QAPI_DSS_CALL_INFO_MAX_E
} qapi_DSS_Call_Param_Identifier_t;
```

- **Parameters**

Parameter	Description
QAPI_DSS_CALL_INFO_UMTS_PROFILE_IDX_E	UMTS profile ID.
QAPI_DSS_CALL_INFO_APN_NAME_E	APN name.
QAPI_DSS_CALL_INFO_USERNAME_E	APN user name.
QAPI_DSS_CALL_INFO_PASSWORD_E	APN password.

QAPI_DSS_CALL_INFO_AUTH_PREF_E	Authentication preference.
QAPI_DSS_CALL_INFO_CDMA_PROFILE_IDX_E	CDMA profile ID.
QAPI_DSS_CALL_INFO_TECH_PREF_E	Technology preference.
QAPI_DSS_CALL_INFO_IP_VERSION_E	Preferred IP family for the call.
QAPI_DSS_CALL_INFO_EXT_TECH_E	Extended technology preference.
QAPI_DSS_CALL_INFO_MO_EXCEPTION_DATA_E	MO exception data.

2.1.1.4. enum qapi_DSS_Net_Evt_t

QAPI DSS event names. Event names are sent along with the registered user callback.

```
typedef enum qapi_DSS_Net_Evt_e
{
   QAPI_DSS_EVT_INVALID_E = 0x00,
   QAPI_DSS_EVT_NET_IS_CONN_E,
   QAPI_DSS_EVT_NET_NO_NET_E,
   QAPI_DSS_EVT_NET_RECONFIGURED_E,
   QAPI_DSS_EVT_NET_NEWADDR_E,
   QAPI_DSS_EVT_NET_DELADDR_E,
   QAPI_DSS_EVT_NIPD_DL_DATA_E,
   QAPI_DSS_EVT_MAX_E
} qapi_DSS_Net_Evt_t;
```

- **Parameters**

Parameter	Description
QAPI_DSS_EVT_INVALID_E	Invalid event.
QAPI_DSS_EVT_NET_IS_CONN_E	Call connected.
QAPI_DSS_EVT_NET_NO_NET_E	Call disconnected.
QAPI_DSS_EVT_NET_RECONFIGURED_E	Call reconfigured.
QAPI_DSS_EVT_NET_NEWADDR_E	New address generated.
QAPI_DSS_EVT_NET_DELADDR_E	Delete generated.
QAPI_DSS_EVT_NIPD_DL_DATA_E	Non-IP downlink data.

2.1.1.5. enum qapi_DSS_IP_Family_t

IP families. This enumeration is used in *qapi_DSS_Get_Call_End_Reason()* function.

```
typedef enum qapi_DSS_IP_Family_e
{
   QAPI_DSS_IP_FAMILY_V4_E = 0x00,
   QAPI_DSS_IP_FAMILY_V6_E,
   QAPI_DSS_NON_IP_FAMILY_E,
   QAPI_DSS_NUM_IP_FAMILIES_E
} qapi_DSS_IP_Family_t;
```

- **Parameters**

Parameter	Description
QAPI_DSS_IP_FAMILY_V4_E	IPV4 address family.
QAPI_DSS_IP_FAMILY_V6_E	IPV6 address family.
QAPI_DSS_NON_IP_FAMILY_E	Non-IP family.

2.1.1.6. enum qapi_DSS_Data_Bearer_Tech_t

Bearer technology types.This enumeration is used in *qapi_DSS_Get_Current_Data_Bearer_Tech()* function.

```
typedef enum qapi_DSS_Data_Bearer_Tech_e
{
   QAPI_DSS_DATA_BEARER_TECH_UNKNOWN_E = 0x00,
   QAPI_DSS_DATA_BEARER_TECH_CDMA_1X_E,
   QAPI_DSS_DATA_BEARER_TECH_EVDO_REV0_E,
   QAPI_DSS_DATA_BEARER_TECH_EVDO_REVA_E,
   QAPI_DSS_DATA_BEARER_TECH_EVDO_REVB_E,
   QAPI_DSS_DATA_BEARER_TECH_EHRPD_E,
   QAPI_DSS_DATA_BEARER_TECH_FMC_E,
   QAPI_DSS_DATA_BEARER_TECH_HRPD_E,
   QAPI_DSS_DATA_BEARER_TECH_3GPP2_WLAN_E,
   QAPI_DSS_DATA_BEARER_TECH_WCDMA_E,
   QAPI_DSS_DATA_BEARER_TECH_GPRS_E,
   QAPI_DSS_DATA_BEARER_TECH_HSDPA_E,
   QAPI_DSS_DATA_BEARER_TECH_HSUPA_E,
   QAPI_DSS_DATA_BEARER_TECH_EDGE_E,
```

```
    QAPI_DSS_DATA_BEARER_TECH_LTE_E,
    QAPI_DSS_DATA_BEARER_TECH_HSDPA_PLUS_E,
    QAPI_DSS_DATA_BEARER_TECH_DC_HSDPA_PLUS_E,
    QAPI_DSS_DATA_BEARER_TECH_HSPA_E,
    QAPI_DSS_DATA_BEARER_TECH_64_QAM_E,
    QAPI_DSS_DATA_BEARER_TECH_TDSCDMA_E,
    QAPI_DSS_DATA_BEARER_TECH_GSM_E,
    QAPI_DSS_DATA_BEARER_TECH_3GPP_WLAN_E,
    QAPI_DSS_DATA_BEARER_TECH_MAX_E
} qapi_DSS_Data_Bearer_Tech_t;
```

- **Parameters**

Parameter	Description
QAPI_DSS_DATA_BEARER_TECH_UNKNOWN_E	Unknown bearer.
QAPI_DSS_DATA_BEARER_TECH_CDMA_1X_E	1X technology.
QAPI_DSS_DATA_BEARER_TECH_EVDO_REV0_E	CDMA Rev 0.
QAPI_DSS_DATA_BEARER_TECH_EVDO_REVA_E	CDMA Rev A.
QAPI_DSS_DATA_BEARER_TECH_EVDO_REVB_E	CDMA Rev B.
QAPI_DSS_DATA_BEARER_TECH_EHRPD_E	EHRPD.
QAPI_DSS_DATA_BEARER_TECH_FMC_E	Fixed mobile convergence.
QAPI_DSS_DATA_BEARER_TECH_HRPD_E	HRPD.
QAPI_DSS_DATA_BEARER_TECH_3GPP2_WLAN_E	IWLAN.
QAPI_DSS_DATA_BEARER_TECH_WCDMA_E	WCDMA.
QAPI_DSS_DATA_BEARER_TECH_GPRS_E	GPRS.
QAPI_DSS_DATA_BEARER_TECH_HSDPA_E	HSDPA.
QAPI_DSS_DATA_BEARER_TECH_HSUPA_E	HSUPA.
QAPI_DSS_DATA_BEARER_TECH_EDGE_E	EDGE.
QAPI_DSS_DATA_BEARER_TECH_LTE_E	LTE.
QAPI_DSS_DATA_BEARER_TECH_HSDPA_PLUS_E	HSDPA+.
QAPI_DSS_DATA_BEARER_TECH_DC_HSDPA_PLUS_E	DC HSDPA+.

QAPI_DSS_DATA_BEARER_TECH_HSPA_E	HSPA.
QAPI_DSS_DATA_BEARER_TECH_64_QAM_E	64 QAM.
QAPI_DSS_DATA_BEARER_TECH_TDSCDMA_E	TD-SCDMA.
QAPI_DSS_DATA_BEARER_TECH_GSM_E	GSM.
QAPI_DSS_DATA_BEARER_TECH_3GPP_WLAN_E	IWLAN.

2.1.1.7. enum qapi_DSS_Call_Tech_Type_t

Call technology. This enumeration is used in *qapi_DSS_Get_Call_Tech()* function.

```
typedef enum qapi_DSS_Call_Tech_Type_e
{
   QAPI_DSS_CALL_TECH_INVALID_E = 0x00,
   QAPI_DSS_CALL_TECH_CDMA_E,
   QAPI_DSS_CALL_TECH_UMTS_E
} qapi_DSS_Call_Tech_Type_t;
```

- **Parameters**

Parameter	Description
QAPI_DSS_CALL_TECH_INVALID_E	Invalid technology.
QAPI_DSS_CALL_TECH_CDMA_E	CDMA.
QAPI_DSS_CALL_TECH_UMTS_E	UMTS.

2.1.2. Definition Type

2.1.2.1. Unique Radio Technology Bitmasks

```
#define QAPI_DSS_RADIO_TECH_UNKNOWN  0x00000000
#define QAPI_DSS_RADIO_TECH_MIN      0x00000001
#define QAPI_DSS_RADIO_TECH_UMTS     QAPI_DSS_RADIO_TECH_MIN
#define QAPI_DSS_RADIO_TECH_CDMA     0x00000002
#define QAPI_DSS_RADIO_TECH_1X       0x00000004
#define QAPI_DSS_RADIO_TECH_DO       0x00000008
#define QAPI_DSS_RADIO_TECH_LTE      0x00000010
```

```
#define QAPI_DSS_RADIO_TECH_TDSCDMA   0x00000020
```

- **Parameters**

Parameter	Description
QAPI_DSS_RADIO_TECH_UNKNOWN	Technology is unknown.
QAPI_DSS_RADIO_TECH_MIN	Start.
QAPI_DSS_RADIO_TECH_UMTS	UMTS.
QAPI_DSS_RADIO_TECH_CDMA	CDMA.
QAPI_DSS_RADIO_TECH_1X	1X.
QAPI_DSS_RADIO_TECH_DO	DO.
QAPI_DSS_RADIO_TECH_LTE	LTE.
QAPI_DSS_RADIO_TECH_TDSCDMA	TDSCDMA.

2.1.2.2. Call Information

```
#define QAPI_DSS_CALL_INFO_USERNAME_MAX_LEN      127
#define QAPI_DSS_CALL_INFO_PASSWORD_MAX_LEN      127
#define QAPI_DSS_CALL_INFO_APN_MAX_LEN           150
#define QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN   12
#define QAPI_DSS_MAX_DATA_CALLS                  20
```

- **Parameters**

Parameter	Description
QAPI_DSS_CALL_INFO_USERNAME_MAX_LEN	Maximum length of the username.
QAPI_DSS_CALL_INFO_PASSWORD_MAX_LEN	Maximum length of the password.
QAPI_DSS_CALL_INFO_APN_MAX_LEN	Maximum length of the APN.
QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN	Maximum length of the device name.
QAPI_DSS_MAX_DATA_CALLS	Maximum Client Handles Supported.

2.1.2.3. QAPI_DSS Error Codes

```
#define QAPI_DSS_SUCCESS      0
#define QAPI_DSS_ERROR        -1
```

- **Parameters**

Parameter	Description
QAPI_DSS_SUCCESS	Indicates that the operation was successful.
QAPI_DSS_ERROR	Indicates that the operation was not successful.

2.1.2.4. IP Versions

```
#define QAPI_DSS_IP_VERSION_4      4
#define QAPI_DSS_IP_VERSION_6      6
#define QAPI_DSS_IP_VERSION_4_6    10
```

- **Parameters**

Parameter	Description
QAPI_DSS_IP_VERSION_4	IP version v4.
QAPI_DSS_IP_VERSION_6	IP version v6.
QAPI_DSS_IP_VERSION_4_6	IP version v4v6.

2.1.3. Typedefs

2.1.3.1. typedef void (*qapi_DSS_Net_Ev_CB_t)(qapi_DSS_Hndl_t hndl, void *user_data,qapi_DSS_Net_Evt_t evt, qapi_DSS_Evt_Payload_t *payload_ptr);

Callback function prototype for DSS events.

- **Prototype**

```
typedef void (*qapi_DSS_Net_Ev_CB_t)
(
   qapi_DSS_Hndl_t              hndl,
```

```
    void                        *user_data,
    qapi_DSS_Net_Evt_t          evt,
    qapi_DSS_Evt_Payload_t *payload_ptr
);
```

- **Parameters**

hndl:
[In] Handle to which this event is associated.

user_data:
[In] Application-provided user data.

evt:
[In] Event identifier.

payload_ptr:
[In] Pointer to associated event information.

- **Return Value**

None.

2.1.4. Stucture Type

2.1.4.1. struct qapi_DSS_CE_Reason_t

Call end (CE) reason.

```
typedef struct qapi_DSS_CE_Reason_s
{
    qapi_DSS_CE_Reason_Type_t reason_type;
    int reason_code;
} qapi_DSS_CE_Reason_t;
```

- **Parameters**

Type	Parameter	Description
qapi_DSS_CE_Reason_Type_t	*reason_type*	Discriminator for reason codes.
int	*reason_code*	Overloaded cause codes discriminated by reason type.

2.1.4.2. struct qapi_DSS_Call_Param_Value_t

Specifies call parameter values.

```
typedef struct qapi_DSS_Call_Param_Value_s
{
   char *buf_val;
   int     num_val;
} qapi_DSS_Call_Param_Value_t;
```

- **Parameters**

Type	Parameter	Description
char *	*buf_val*	Pointer to the buffer containing the parameter value that is to be set.
int	*num_val*	Size of the parameter buffer.

2.1.4.3. struct qapi_DSS_Addr_t

Structure to represent the IP address.

```
typedef struct qapi_DSS_Addr_s
{
   char valid_addr;
   union qapi_dss_ip_address_u
   {
      uint32_t v4;
      uint64_t v6_addr64[2];
      uint32_t v6_addr32[4];
      uint16_t v6_addr16[8];
      uint8_t   v6_addr8[16];
   } addr;
} qapi_DSS_Addr_t;
```

- **Parameters**

Type	Parameter	Description
char	*valid_addr*	Indicates whether a valid address is available.
union qapi_dss_ip_address_u	*addr*	Union of DSS IP addresses.

2.1.4.4. union qapi_DSS_Addr_t::qapi_dss_ip_address_u

Union of DSS IP addresses.

```
union qapi_dss_ip_address_u
  {
    uint32_t v4;
    uint64_t v6_addr64[2];
    uint32_t v6_addr32[4];
    uint16_t v6_addr16[8];
    uint8_t   v6_addr8[16];
  } addr;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*v4*	Used to access the IPv4 address.
uint64_t	*v6_addr64*	Used to access the IPv6 address.
uint32_t	*v6_addr32*	Used to access the IPv6 address as four 32-bit integers.
uint16_t	*v6_addr16*	Used to access octets of the IPv6 address.
uint8_t	*v6_addr8*	Used to access octets of the IPv6 address as 16 8-bit integers.

2.1.4.5. struct qapi_DSS_Addr_Info_t

IP address-related information.

```
typedef struct qapi_DSS_Addr_Info_s
{
  qapi_DSS_Addr_t    iface_addr_s;
  unsigned int         iface_mask;
  qapi_DSS_Addr_t    gtwy_addr_s;
  unsigned int         gtwy_mask;
  qapi_DSS_Addr_t    dnsp_addr_s;
  qapi_DSS_Addr_t    dnss_addr_s;
} qapi_DSS_Addr_Info_t;
```

- **Parameters**

Type	Parameter	Description
qapi_DSS_Addr_t	*iface_addr_s*	Network interface address.
unsigned int	*iface_mask*	Interface subnet mask.
qapi_DSS_Addr_t	*gtwy_addr_s*	Gateway server address.
unsigned int	*gtwy_mask*	Gateway subnet mask.
qapi_DSS_Addr_t	*dnsp_addr_s*	Primary DNS server address.
qapi_DSS_Addr_t	*dnss_addr_s*	Secondary DNS server address.

2.1.4.6. struct qapi_DSS_Data_Pkt_Stats_t

Packet statistics.

```
typedef struct qapi_DSS_Data_Pkt_Stats_s
{
   unsigned long pkts_tx;
   unsigned long pkts_rx;
   long long     bytes_tx;
   long long     bytes_rx;
   unsigned long pkts_dropped_tx;
   unsigned long pkts_dropped_rx;
} qapi_DSS_Data_Pkt_Stats_t
```

- **Parameters**

Type	Parameter	Description
unsigned long	*pkts_tx*	Number of packets transmitted.
unsigned long	*pkts_rx*	Number of packets received.
long long	*bytes_tx*	Number of bytes transmitted.
long long	*bytes_rx*	Number of bytes received.
unsigned long	*pkts_dropped_tx*	Number of transmit packets dropped.
unsigned long	*pkts_dropped_rx*	Number of receive packets dropped.

2.1.4.7. struct qapi_DSS_Evt_Payload_t

Event payload sent with event callbacks.

```
typedef struct qapi_DSS_Evt_Payload_s
{
   uint8_t *data;
   uint32_t data_len;
} qapi_DSS_Evt_Payload_t
```

- **Paramters**

Type	Parameter	Description
uint8_t *	*data*	Payload data.
uint32_t	*data_len*	Payload data length.

2.2. API Functions

2.2.1. qapi_DSS_Init

Initializes the DSS netctrl library for the specified operating mode. This function must be invoked once per process, typically on process startup.

NOTE

Only *QAPI_DSS_MODE_GENERAL* is to be used by applications.

- **Prototype**

```
qapi_Status_t qapi_DSS_Init(int mode);
```

- **Parameters**

mode:
[In] Mode of operation in which to initialize the library.

- **Return Value**

QAPI_OK This function is executed successfully.

QAPI_ERROR It fails to execute this function.

- **Dependencies**

None.

2.2.2. qapi_DSS_Release

Cleans up the DSS netctrl library. This function must be invoked once per process, typically to clean up the resources at the end.

NOTE

Only *QAPI_DSS_MODE_GENERAL* is to be used by applications.

- **Prototype**

```
qapi_Status_t qapi_DSS_Release(int mode);
```

- **Parameters**

mode:
[In] Mode of operation in which to initialize the library.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

None.

2.2.3. qapi_DSS_Get_Data_Srvc_Hndl

Gets an opaque data service handle. All subsequent functions use this handle as an input parameter.

NOTE

DSS netctrl library waits for initialization from the lower layers (QMI ports being opened, the RmNet interfaces being available, etc.) to support data services functionality. During initial bootup scenarios, these dependencies may not be available, which will cause an error to be returned by *dss_get_data_srvc_hndl*. In such cases, clients are asked to retry this function call repeatedly using a 500 ms timeout interval. Once a non-NULL handle is returned, clients can exit out of the delayed retry loop.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Data_Srvc_Hndl (qapi_DSS_Net_Ev_CB_t user_cb_fn, void * user_data, qapi_DSS_Hndl_t * hndl );
```

- **Parameters**

user_cb_fun:
[In] Client callback function used to post event indications.

user_data:
[In] Pointer to the client context block (cookie). The value may be NULL.

hndl:
[In] Pointer to data service handle.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

qapi_DSS_Init() must have been called first.

2.2.4. qapi_DSS_Rel_Data_Srvc_Hndl

Releases a data service handle. All resources associated with the handle in the library are released.

NOTE

If the user starts an interface with this handle, the corresponding interface is stopped before the DSS handle is released.

- **Prototype**

```
qapi_Status_t qapi_DSS_Rel_Data_Srvc_Hndl ( qapi_DSS_Hndl_t hndl );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

- **Return Value**

QAPI_OK	This function is executed successfully..

QAPI_ERROR It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.5. qapi_DSS_Set_Data_Call_Param

Sets the data call parameter before trying to start a data call. Clients may call this function multiple times with various types of parameters that need to be set.

- **Prototype**

```
qapi_Status_t qapi_DSS_Set_Data_Call_Param ( qapi_DSS_Hndl_t hndl, qapi_DSS_Call_Param_Identifier_t identifier, qapi_DSS_Call_Param_Value_t *info );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

identifier:
[In] Identifies the parameter information.

info:
[In] Parameter value that is to be set.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.6. qapi_DSS_Start_Data_Call

Starts a data call.

An immediate call return value indicates whether the request was sent successfully. The client receives asynchronous notifications via a callback registered with *qapi_DSS_Get_Data_Srvc_Hndl()* indicating the data call bring-up status.

- **Prototype**

```
qapi_Status_t qapi_DSS_Start_Data_Call ( qapi_DSS_Hndl_t hndl );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

- **Return Value**

QAPI_OK this function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.7. qapi_DSS_Stop_Data_Call

Stops a data call.

An immediate call return value indicates whether the request was sent successfully. The client receives asynchronous notification via a callback registered with *qapi_DSS_Get_Data_Srvc_Hndl()* indicating the data call tear-down status.

- **Prototype**

```
qapi_Status_t qapi_DSS_Stop_Data_Call ( qapi_DSS_Hndl_t hndl );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.
The data call must have been brought up using *qapi_DSS_Start_Data_Call()*.

2.2.8. qapi_DSS_Get_Pkt_Stats

Queries the packet data transfer statistics from the current packet data session.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Pkt_Stats ( qapi_DSS_Hndl_t hndl, qapi_DSS_Data_Pkt_Stats_t* dss_data_stats );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

dss_data_stats:
[In] Buffer to hold the queried statistics details.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.9. qapi_DSS_Reset_Pkt_Stats

Resets the packet data transfer statistics.

- **Prototype**

```
qapi_Status_t qapi_DSS_Reset_Pkt_Stats ( qapi_DSS_Hndl_t hndl );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.10. qapi_DSS_Get_Call_End_Reason

Queries for the reason a data call was ended.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Call_End_Reason ( qapi_DSS_Hndl_t hndl,qapi_DSS_CE_Reason_t * ce_reason, qapi_DSS_IP_Family_t ip_family );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

ce_reason:
[Out] Buffer to hold data call ending reason information.

ip_family:
[In] IP family for which the call end reason was requested.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.11. qapi_DSS_Get_Call_Tech

Gets the techcology on which the call was brought up. This function can be called any time after the client receives the *QAPI_DSS_EVT_NET_IS_CONN* event and before the client releases the DSS handle.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Call_Tech ( qapi_DSS_Hndl_t hndl, qapi_DSS_Call_Tech_Type_t * call_tech );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

call_tech:
[Out] Buffer to hold the call technology.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.12. qapi_DSS_Get_Current_Data_Bearer_Tech

Queries the data bearer techcology on which the call was brought up. This function can be called any time after *QAPI_DSS_EVT_NET_IS_CONN* event is received by the client and before the client releases the dss handle.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Current_Data_Bearer_Tech ( qapi_DSS_Hndl_t hndl,
qapi_DSS_Data_Bearer_Tech_t * bearer_tech );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

bearer_tech:
[In] Pointer to where to retrieve the data bearer technology.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.13. qapi_DSS_Get_Device_Name

Queries the data interface name for the data call associated with the specified data service handle.

NOTE

len must be at least *QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN* + 1 long.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Device_Name ( qapi_DSS_Hndl_t hndl, char* buf, int len );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

buf:
[Out] Buffer to hold the data interface name string.

len:
[In] Length of the buffer allocated by the client.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.14. qapi_DSS_Get_Qmi_Port_Name

Queries the QMI port name for the data call associated with the specified data service handle.

NOTE

len must be at least *QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN* + 1 long.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Qmi_Port_Name ( qapi_DSS_Hndl_t hndl, char* buf, int len );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

buf:
[Out] Buffer to hold the QMI port name string.

len:
[In] Length of the buffer allocated by the client.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It is fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.15. qapi_DSS_Get_IP_Addr_Count

Queries the number of IP addresses (IPv4 and global IPv6) associated with the DSS interface.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_IP_Addr_Count ( qapi_DSS_Hndl_t hndl, unsigned int * ip_addr_cnt );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

ip_addr_cnt:
[In] Pointer to where to retrieve the number of IP addresses associated with the DSS interface.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It is fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

2.2.16. qapi_DSS_Get_IP_Addr

Queries the IP address information structure (network order).

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_IP_Addr ( qapi_DSS_Hndl_t hndl, qapi_DSS_Addr_Info_t * info_ptr, int len );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

info_ptr:
[Out] Buffer containing the IP address information.

len:
[In] Number of IP address buffers.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It is fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.
The length parameter can be obtained by calling *qapi_DSS_Get_IP_Addr_Count()*.
It is assumed that the client has allocated memory for enough structures specified by the len field.

2.2.17. qapi_DSS_Get_IP_Addr_Per_Family

Queries the IP address information structure (network order).

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_IP_Addr_Per_Family ( qapi_DSS_Hndl_t hndl, qapi_DSS_Addr_Info_t * info_ptr, unsigned int addr_family );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

info_ptr:
[Out] Buffer containing the IP address information.

addr_family:
[In] IPV4/IPV6.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It is fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.
The length parameter can be obtained by calling *qapi_DSS_Get_IP_Addr_Count()*.
It is assumed that the client has allocated memory for enough structures specified by the len field.

2.2.18. qapi_DSS_Get_Link_Mtu

Queries the MTU information associated with the link.

- **Prototype**

```
qapi_Status_t qapi_DSS_Get_Link_Mtu ( qapi_DSS_Hndl_t hndl, unsigned int * mtu );
```

- **Parameters**

hndl:
[In] Handle received from *qapi_DSS_Get_Data_Srvc_Hndl()*.

mtu:
[Out] Buffer containing the MTU information.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

qapi_DSS_init() must have been called first.
A valid handle must be obtained by *qapi_DSS_Get_Data_Srvc_Hndl()*.

3 Network Socket APIs

The QAPI network socket API is a collection of standard functions that allow the application to include Internet communications capabilities. The sockets are based on the Berkeley Software Distribution (BSD) sockets. In general, the BSD socket interface relies on Client-Server architecture and uses a socket object for every operation. The interface supports TCP (*SOCK_STREAM*) and UDP (*SOCK_DGRAM*), Server mode and Client mode, as well as IPv4 and IPv6 communication.

A socket can be configured with specific options (see Socket Options). Due to the memory-constrained properties of the device, it is mandatory to follow the BSD socket programming guidelines, and in particular, check for return values of each function. There is a chance that an operation may fail due to resource limitations. For example, the send function may be able to send only some of the data and not all of it in a single call. A subsequent call with the rest of the data is then required. In some other cases, an application thread may need to sleep in order to allow the system to clear its queues, process data, and so on.

This chapter provides the following APIs:

qapi_socket
qapi_bind
qapi_listen
qapi_accept
qapi_connect
qapi_setsockopt
qapi_getsockopt
qapi_socketclose
qapi_errno
qapi_recvfrom
qapi_recv
qapi_sendto
qapi_send
qapi_select
qapi_fd_zero
qapi_fd_clr
qapi_fd_isset
qapi_getpeername
qapi_getsockname

3.1. Data Types

3.1.1. Definition Type

3.1.1.1. Address Families

```
#define  AF_UNSPEC       0
#define  AF_INET         2
#define  AF_INET6        3
#define  AF_INET_DUAL46  4
```

- **Parameters**

Parameter	Description
AF_UNSPEC	Address family is unspecified.
AF_INET	Address family is IPv4.
AF_INET6	Address family is IPv6.
AF_INET_DUAL46	Address family is IPv4 and IPv6.

3.1.1.2. Socket Types

```
#define  SOCK_STREAM  1
#define  SOCK_DGRAM   2
#define  SOCK_RAW     3
```

- **Parameters**

Parameter	Description
SOCK_STREAM	Socket stream (TCP).
SOCK_DGRAM	Socket datagram (UDP).
SOCK_RAW	Raw socket.

3.1.1.3. BSD Socket Error Codes

```
#define ENOBUFS  1
```

```
#define ETIMEDOUT       2
#define EISCONN         3
#define EOPNOTSUPP      4
#define ECONNABORTED    5
#define EWOULDBLOCK     6
#define ECONNREFUSED    7
#define ECONNRESET      8
#define EBADF           9
#define EALREADY        10
#define EINVAL          11
#define EMSGSIZE        12
#define EPIPE           13
#define EDESTADDRREQ    14
#define ESHUTDOWN       15
#define ENOPROTOOPT     16
#define EHAVEOOB        17
#define ENOMEM          18
#define EADDRNOTAVAIL   19
#define EADDRINUSE      20
#define EAFNOSUPPORT    21
#define EINPROGRESS     22
#define ELOWER          23
#define ENOTSOCK        24
#define EIEIO           27
#define ETOOMANYREFS    28
#define EFAULT          29
#define ENETUNREACH     30
#define ENOTCONN        31
```

- **Parameters**

Parameter	Description
ENOBUFS	No buffer space is available.
ETIMEDOUT	Operation timed out.
EISCONN	Socket is already connected.
EOPNOTSUPP	Operation is not supported.
ECONNABORTED	Software caused a connection abort.
EWOULDBLOCK	Socket is marked nonblocking and the requested operation will block.
ECONNREFUSED	Connection was refused.

ECONNRESET	Connection was reset by peer.
EBADF	An invalid descriptor was specified.
EALREADY	Operation is already in progress.
EINVAL	Invalid argument was passed.
EMSGSIZE	Message is too long.
EPIPE	The local end has been shut down on a connection-oriented socket.
EDESTADDRREQ	Destination address is required.
ESHUTDOWN	Cannot send after a socket shutdown.
ENOPROTOOPT	Protocol is not available.
EHAVEOOB	Out of band
ENOMEM	No memory is available.
EADDRNOTAVAIL	Cannot assign the requested address.
EADDRINUSE	Address is already in use.
EAFNOSUPPORT	Address family is not supported by the protocol family.
EINPROGRESS	Operation is in progress.
ELOWER	Lower layer (IP) error.
ENOTSOCK	Socket operation on nonsocket.
EIEIO	I/O error.
ETOOMANYREFS	Too many references.
EFAULT	Bad address.
ENETUNREACH	Network is unreachable.
ENOTCONN	Socket is not connected.

3.1.1.4. Socket Options

```
#define    SOL_SOCKET          -1
#define    SO_ACCEPTCONN       0x00002
```

```
#define SO_REUSEADDR 0x00004
#define SO_KEEPALIVE 0x00008
#define SO_DONTROUTE 0x00010
#define SO_BROADCAST 0x00020
#define SO_USELOOPBACK 0x00040
#define SO_LINGER 0x00080
#define SO_OOBINLINE 0x00100
#define SO_TCPSACK 0x00200
#define SO_WINSCALE 0x00400
#define SO_TIMESTAMP 0x00800
#define SO_BIGCWND 0x01000
#define SO_HDRINCL 0x02000
#define SO_NOSLOWSTART 0x04000
#define SO_FULLMSS 0x08000
#define SO_SNDTIMEO 0x1005
#define SO_RCVTIMEO 0x1006
#define SO_ERROR 0x1007
#define SO_RXDATA 0x1011
#define SO_TXDATA 0x1012
#define SO_MYADDR 0x1013
#define SO_NBIO 0x1014
#define SO_BIO 0x1015
#define SO_NONBLOCK 0x1016
#define SO_CALLBACK 0x1017
#define SO_UDPCALLBACK 0x1019
#define IPPROTO_IP 0
#define IP_HDRINCL 2
#define IP_MULTICAST_IF 9
#define IP_MULTICAST_TTL 10
#define IP_MULTICAST_LOOP 11
#define IP_ADD_MEMBERSHIP 12
#define IP_DROP_MEMBERSHIP 13
#define IPV6_MULTICAST_IF 80
#define IPV6_MULTICAST_HOPS 81
#define IPV6_MULTICAST_LOOP 82
#define IPV6_JOIN_GROUP 83
#define IPV6_LEAVE_GROUP 84
#define IP_EXCLUDE_LIST 17
#define IP_OPTION 1
#define IP_TOS 3
#define IP_TTL_OPT 4
#define IPV6_SCOPEID 14
#define IPV6_UNICAST_HOPS 15
#define IPV6_TCLASS 16
```

- **Parameters**

Parameter	Description
SOL_SOCKET	For use with *qapi_setsockopt ()* at the socket level.
SO_ACCEPTCONN	Socket has had *listen()*.
SO_REUSEADDR	Allow local address reuse.
SO_KEEPALIVE	Keep connections alive.
SO_DONTROUTE	Not used.
SO_BROADCAST	Not used.
SO_USELOOPBACK	Not used.
SO_LINGER	Linger on close if data is present.
SO_OOBINLINE	Leave the received OOB data in line.
SO_TCPSACK	Allow TCP SACK (selective acknowledgment).
SO_WINSCALE	Set the scaling window option.
SO_TIMESTAMP	Set the TCP timestamp option.
SO_BIGCWND	Large initial TCP congestion window.
SO_HDRINCL	User access to IP header for SOCK_RAW.
SO_NOSLOWSTART	Suppress slowstart on this socket.
SO_FULLMSS	Not used.
SO_SNDTIMEO	Send a timeout.
SO_RCVTIMEO	Receive a timeout.
SO_ERROR	Socket error.
SO_RXDATA	Get a count of bytes in sb_rcv.
SO_TXDATA	Get a count of bytes in sb_snd.
SO_MYADDR	Return my IP address.
SO_NBIO	Set socket to Nonblocking mode.

SO_BIO	Set socket to Blocking mode.
SO_NONBLOCK	Set/get blocking mode via the optval parameter.
SO_CALLBACK	Set/get the TCP zero_copy callback routine.
SO_UDPCALLBACK	Set/get the UDP zero_copy callback routine.
IPPROTO_IP	For use with *qapi_setsockopt()* at IPPROTO_IP level.
IP_HDRINCL	IP header is included with the data.
IP_MULTICAST_IF	Set/get the IP multicast interface.
IP_MULTICAST_TTL	Set/get the IP multicast TTL.
IP_MULTICAST_LOOP	Set/get the IP multicast loopback.
IP_ADD_MEMBERSHIP	Add an IPv4 group membership.
IP_DROP_MEMBERSHIP	Drop an IPv4 group membership.
IPV6_MULTICAST_IF	Set the egress interface for multicast traffic.
IPV6_MULTICAST_HOPS	Set the number of hops.
IPV6_MULTICAST_LOOP	Enable/disable loopback for multicast.
IPV6_JOIN_GROUP	Join an IPv6 MC group.
IPV6_LEAVE_GROUP	Leave an IPv6 MC group.
IP_EXCLUDE_LIST	Set/get the exclude list for 255 RAW socket.
IP_OPTIONS	For use with *qapi_setsockopt()* at IP_OPTIONS level.
IP_TOS	IPv4 type of service and precedence.
IP_TTL_OPT	IPv4 time to live.
IPV6_SCOPEID	IPv6 IF scope ID.
IPV6_UNICAST_HOPS	IPv6 hop limit.
IPV6_TCLASS	IPv6 traffic class.

3.1.1.5. Flags for recv() and send()

```
#define     MSG_OOB             0x1
#define     MSG_PEEK            0x2
#define     MSG_DONTROUTE       0x4
#define     MSG_DONTWAIT        0x20
#define     MSG_ZEROCOPYSEND    0x1000
```

- **Parameters**

Parameter	Description
MSG_OOB	Send/receive out-of-band data.
MSG_PEEK	Peek at the incoming message.
MSG_DONTROUTE	Send without using routing tables.
MSG_DONTWAIT	Send/receive is nonblocking.
MSG_ZEROCOPYSEND	Send with zero-copy.

3.1.1.6. Infinite Time for the timeout_ms Argument in qapi_select().

```
#define QAPI_NET_WAIT_FOREVER (0xFFFFFFFF)
```

3.1.1.7. Macros to Manipulate fd_set

```
#define FD_ZERO(set)            qapi_fd_zero((set))
#define FD_CLR(handle, set)     qapi_fd_clr((handle), (set))
#define FD_SET(handle, set)     qapi_fd_set((handle), (set))
#define FD_ISSET(handle, set)   qapi_fd_isset((handle), (set))
```

- **Parameters**

Parameter	Description
FD_ZERO(set)	Clears a set.
FD_CLR(handle, set)	Removes a given file descriptor from a set.
FD_SET(handle, set)	Adds a given file descriptor from a set.

FD_ISSET(handle, set)	Tests to see if a file descriptor is part of the set after *select()* returns.

3.1.2. Structure Type

3.1.2.1. struct in_addr

IPv4 Internet address.

```
struct in_addr
{
    uint32_t s_addr;
};
```

- **Parameters**

Type	Parameter	Description
uint32_t	*s_addr*	IPv4 address in network order.

3.1.2.2. struct sockaddr_in

BSD-style socket IPv4 Internet address.

```
struct sockaddr_in
{
    uint16_t        sin_family;
    uint16_t        sin_port;
    struct in_addr  sin_addr;
    uint8_t         sin_zero[8];
};
```

- **Parameters**

Type	Parameter	Description
uint16_t	*sin_family*	*AF_INET*
uint16_t	*sin_port*	UDP/TCP port number in network order.
struct in_addr	*sin_addr*	IPv4 address in network order.

uint8_t	*sin_zero*	Reserved – must be zero.

3.1.2.3. struct in6_addr

IPv6 Internet address.

```
typedef struct in6_addr
{
    uint8_t   s_addr[16];
} ip6_addr;
```

- **Parameters**

Type	Parameter	Description
uint8_t	*s_addr*	128-bit IPv6 address.

3.1.2.4. struct ip46addr_n

BSD-style socket IPv6 Internet address.

```
struct ip46addr_n
{
    uint16_t type;
    union
    {
        unsigned long     addr4;
        uint8_t           addr6[16];
    } a;
    union
    {
        unsigned long     gtwy4;
        uint8_t           gtwy6[16];
    } g;
    uint32_t subnet;
};
```

- **Parameters**

Type	Parameter	Description
uint16_t	*s_addr*	*AF_INET* or *AF_INET6*.
union ip46addr_n	*a*	Address union.
union ip46addr_n	*g*	Gateway union.
uint32_t	*subnet*	Subnet.

3.1.2.5. union ip46addr_n.a

```
union
{
    unsigned long    addr4;
    uint8_t          addr6[16];
} a;
```

- **Parameters**

Type	Parameter	Description
unsigned long	*addr4*	IPv4 address.
uint8_t	*addr6*	IPv6 address.

3.1.2.6. union ip46addr_n.g

```
union
{
    unsigned long    gtwy4;
    uint8_t          gtwy6[16];
} g;
```

- **Parameters**

Type	Parameter	Description
unsigned long	*addr4*	IPv4 address.
uint8_t	*addr6*	IPv6 address.

3.1.2.7. struct sockaddr_in6

Socket address information.

```
struct sockaddr_in6
{
    uint16_t        sin_family;
    uint16_t        sin_port;
    uint32_t        sin_flowinfo;
    struct in6_addr sin_addr;
    int32_t         sin_scope_id;
};
```

- **Parameters**

Type	Parameter	Description
uint16_t	*sin_family*	*AF_INET6*.
uint16_t	*sin_port*	UDP/TCP port number in network order.
uint32_t	*sin_flowinfo*	IPv6 flow information.
struct in6_addr	*sin_addr*	IPv6 address.
int32_t	*sin_scope_id*	Set of interfaces for a scope.

3.1.2.8. struct ip46addr

Socket IPv4/IPv6 Internet address union.

```
struct ip46addr
{
    uint16_t type;
    union
    {
        unsigned long   addr4;
        ip6_addr        addr6;
    } a;
};
```

- **Parameters**

Type	Parameter	Description
uint16_t	*type*	*AF_INET* or *AF_INET6*.
union ip46addr	*a*	Address union.

3.1.2.9. union ip46addr.a

```
union
{
    unsigned long    addr4;
    ip6_addr         addr6;
} a;
```

- **Parameters**

Type	Parameter	Description
unsigned long	*addr4*	IPv4 address.
ip6_addr	*addr6*	IPv6 address.

3.1.2.10. struct sockaddr

Generic socket Internet address.

```
struct sockaddr
{
    uint16_t sa_family;
    uint16_t sa_port;
    uint8_t  sa_data[32];
};
```

- **Parameters**

Type	Parameter	Description
uint16_t	*sa_family*	Address family.
uint16_t	*sa_port*	Port number in network order.

uint8_t	*sa_data*	Big enough for 16-byte IPv6 address.

3.1.2.11. struct fd_set

File descriptor sets for *qapi_select()*.

```
typedef struct
{
    uint32_t fd_count;
    uint32_t fd_array[FD_SETSIZE];
} fd_set;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*fd_count*	File descriptor count.
uint32_t	*fd_array*	File descriptor array.

3.2. API Functions

3.2.1. qapi_socket

Creates an endpoint for communication.

- **Prototype**

```
int qapi_socket ( int32_t family, int32_t type, int32_t protocol );
```

- **Parameters**

family:
[In] Protocol family used for communication. The supported families are:
AF_INET IPv4 Internet protocols
AF_INET6 IPv6 Internet protocols

type:
[In] Transport mechanism used for communication. The supported types are:
SOCK_STREAM TCP
SOCK_DGRAM UDP

protocol:
[In] Must be set to 0.

- **Return Value**

On success, a handle for the new socket is returned.
On error, -1 is returned.

- **Dependencies**

None.

3.2.2. qapi_bind

Assigns an address to the socket created by *qapi_socket()*.

- **Prototype**

```
qapi_Status_t qapi_bind ( int32_t handle, struct sockaddr * addr, int32_t addrlen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

addr:
[In] Pointer to an address to be assigned to the socket. The actual address structure passed for the *addr* argument will depend on the address family.

addrlen:
[In] Specifies the size, in bytes, of the address pointed to by *addr*.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.3. qapi_listen

Marks the socket as a passive socket.

- **Prototype**

```
qapi_Status_t qapi_listen ( int32_t handle, int32_t backlog );
```

- **Parameters**

handle:
[In] Handle (returned from *qapi_socket()*) that refers to a *SOCK_STREAM* socket.

backlog:
[In] Define the maximum length to which the queue of pending connections for the handle may grow.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.4. qapi_accept

Accepts a connection request from the peer on a *SOCK_STREAM* socket.

This function is used with a *SOCK_STREAM* socket. It extracts the first connection request on the queue of pending connections for the listening socket (i.e., *handle*), creates a new connected socket, and returns a new socket handle referring to that socket. The newly created socket is in the Established state. The original socket (i.e., *handle*) is unaffected by this call. If no pending connections are present on the queue, and the socket is not marked as nonblocking, *qapi_accept()* blocks the caller until a connection is present. If the socket is marked nonblocking and no pending connections are present on the queue, *qapi_accept()* fails with the error *EAGAIN* or *EWOULDBLOCK*.

- **Prototype**

```
int qapi_accept ( int32_t handle, struct sockaddr * cliaddr, int32_t * addrlen );
```

- **Parameters**

handle:
[In] Socket handle that has been created with *qapi_socket()*, bound to a local address with *qapi_bind()*, and listens for connections after *qapi_listen()*.

cliaddr:
[Out] Pointer to a *sockaddr* structure. This structure is filled in with the address of the peer socket. The

exact format of the address returned (i.e., *cliaddr*) is determined by the socket's address family. When *cliaddr* is NULL, nothing is filled in; in this case, *addrlen* should also be NULL.

addrlen:
[Out] Value-result argument: The caller must initialize it to contain the size (in bytes) of the structure pointed to by *cliaddr*. On return, it will contain the actual size of the peer address.

- **Return Value**

On success, the call returns a positive integer that is a handle for the accepted socket.
On error, -1 is returned.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*, bound to a local address with *qapi_bind()*, and listens for connections after *qapi_listen()*.

3.2.5. qapi_connect

Initiates a connection on a socket.

If the socket is of type *SOCK_DGRAM, *svraddr* is the address to which datagrams are sent by default, and the only address from which datagrams are received. If the socket is of type *SOCK_STREAM*, this call attempts to make a connection to the socket that is bound to the address specified by *srvaddr*.

- **Prototype**

```
qapi_Status_t qapi_connect ( int32_t handle, struct sockaddr * srvaddr, int32_t addrlen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

srvaddr:
[In] Pointer to the peer's address to which the socket is connected.

addrlen:
[In] Specify the size (in bytes) of *srvaddr*.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.6. qapi_setsockopt

Sets the options for a socket.

- **Prototype**

```
qapi_Status_t qapi_setsockopt ( int32_t handle, int32_t level, int32_t optname, void * optval, int32_t optlen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

level:
[In] Protocol level at which the option exists.

optname:
[In] Name of the option.

optval:
[In] Pointer to the option value to be set.

optlen:
[In] Option length in bytes.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.7. qapi_getsockopt

Gets the options for a socket.

- **Prototype**

```
qapi_Status_t qapi_getsockopt ( int32_t handle, int32_t level, int32_t optname, void * optval, int32_t optlen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

level:
[In] Protocol level at which the option exists.

optname:
[In] Name of the option.

optval:
[Out] Pointer to a buffer in which the value for the requested option is to be returned.

optlen:
[In] Option length in bytes.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.8. qapi_socketclose

Closes a socket.

- **Prototype**

```
qapi_Status_t qapi_socketclose ( int32_t handle );
```

- **Parameters**

handle:
[in] Socket handle returned from *qapi_socket()*.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.9. qapi_errno

Gets the last error code on a socket.

- **Prototype**

```
int qapi_errno ( int32_t handle );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

- **Return Value**

Socket error code or *ENOTSOCK* if socket is not found.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.10. qapi_recvfrom

Receives a message from a socket.

- **Prototype**

```
int qapi_recvfrom ( int32_t handle, char * buf, int32_t len, int32_t flags, struct sockaddr * from, int32_t * fromlen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

buf:
[Out] Pointer to a buffer for the received message.

len:
[In] Number of bytes to receive.

flags:
[In] 0, or it is formed by ORing one or more of:

MSG_PEEK	Causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data.
MSG_OOB	Requests receipt of out-of-band data that would not be received in the normal data stream.
MSG_DONTWAIT	Enables a nonblocking operation; if the operation blocks, the call fails with the error *EAGAIN* or *EWOULDBLOCK*.

from:
[Out] If not NULL, and the underlying protocol provides the source address, this source address is filled in. When NULL, nothing is filled in, in this case, fromlen is not used, and should also be NULL

fromlen:
[Out] This is a value-result argument, which the caller should initialize before the call to the size of the buffer associated with from, and modified on return to indicate the actual size of the source address.

- **Return Value**

On success, the number of bytes received is returned.
On error, -1 is returned.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.11. qapi_recv

Receives a message from a socket.

The *qapi_recv()* call is normally used only on a connected socket and is identical to *qapi_recvfrom(handle, buf, len, flags, NULL, NULL)*.

- **Prototype**

```
int qapi_recv ( int32_t handle, char * buf, int32_t len, int32_t flags );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

buf:
[Out] Pointer to a buffer for the received message.

len:
[In] Number of bytes to receive.

flags:
[In] 0, or it is formed by ORing one or more of:

MSG_PEEK	Causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data.
MSG_OOB	Requests receipt of out-of-band data that would not be received in the normal data stream.
MSG_DONTWAIT	Enables a nonblocking operation; if the operation blocks, the call fails with the error *EAGAIN* or *EWOULDBLOCK*.

- **Return Value**

On success, the number of bytes received is returned.
On error, -1 is returned.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*, and the socket must be on a connected state.

3.2.12. qapi_sendto

Sends a message on a socket to a target.

- **Prototype**

```
int qapi_sendto ( int32_t handle, char * buf, int32_t len, int32_t flags, struct sockaddr * to, int32_t tolen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

buf:
[In] Pointer to a buffer containing the message to be sent.

len:
[In] Number of bytes to send.

flags:
[In] 0, or it is formed by ORing one or more of:

MSG_OOB	Sends out-of-band data on sockets that support this notion (e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data.
MSG_DONTWAIT	Enables a nonblocking operation; if the operation blocks, the call fails with the error *EAGAIN* or *EWOULDBLOCK*.
MSG_DONTROUTE	Don not use a gateway to send the packet; only send it to hosts on directly-connected networks. This is usually used only by diagnostic or routing programs.

to:
[In] Pointer to the address of the target.

tolen:
[In] Size in bytes of the target address.

- **Return Value**

On success, the number of bytes sent is returned.
On error, -1 is returned and errno is set appropriately.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.13. qapi_send

Sends a message on a socket.

The call may be used only when the socket is in a connected state (so that the intended recipient is known). It is equivalent to *qapi_sendto(handle, buf, len, flags, NULL, 0)*.

- **Prototype**

```
qapi_send(int32_t handle, char *buf, int32_t len, int32_t flags)
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

buf:
[In] Pointer to a buffer containing the message to be sent.

len:
[In] Number of bytes to send.

flags:
[In] 0, or it is formed by ORing one or more of:

MSG_OOB	Sends out-of-band data on sockets that support this notion (e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data.
MSG_DONTWAIT	Enables a nonblocking operation; if the operation blocks, the call fails with the error *EAGAIN* or *EWOULDBLOCK*.
MSG_DONTROUTE	Don not use a gateway to send the packet; only send it to hosts on directly-connected networks. This is usually used only by diagnostic or routing programs.

- **Return Value**

On success, the number of bytes sent is returned.
On error, -1 is returned and errno is set appropriately.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*, and the socket must be on a connected state.

3.2.14. qapi_select

Monitors multiple socket handles, waiting until one or more of them become "ready" for some class of I/O operation (e.g., read, write, etc.).

The call causes the calling process to block waiting for activity on any of a list of sockets. Arrays of socket handles are passed for read, write, and exception events. A timeout in milliseconds is also passed.

- **Prototype**

```
int qapi_select ( fd_set * rd, fd_set * wr, fd_set * ex, int32_t timeout_ms );
```

- **Parameters**

rd:
[In] Pointer to a list of read socket handles.

wr:
[In] Pointer to a list of write socket handles.

ex:
[In] Pointer to a list of exception socket handles.

timeout_ms:
[In] Timeout values in milliseconds.

- **Return Value**

The number of sockets that had an event occur and became ready.

- **Dependencies**

A socket that is set to zero must be initialized.

3.2.15. qapi_fd_zero

Initializes a socket that is set to zero.

- **Prototype**

```
qapi_Status_t qapi_fd_zero ( fd_set * set );
```

- **Parameters**

set:
[In] Pointer to a list of sockets.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A socket that is set to zero must be initialized.

3.2.16. qapi_fd_clr

Removes a socket from the socket set.

- **Prototype**

```
qapi_Status_t qapi_fd_clr ( int32_t handle, fd_set * set );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

set:
[In] Pointer to a list of sockets.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.17. qapi_fd_set

Adds a socket to the socket set.

- **Prototype**

```
qapi_Status_t qapi_fd_set ( int32_t handle, fd_set * set );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

set:
[In] Pointer to a list of sockets.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.18. qapi_fd_isset

Checks whether a socket is a member of a socket set.

- **Prototype**

```
qapi_Status_t qapi_fd_isset ( int32_t handle, fd_set * set );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

set:
[In] Pointer to a list of sockets.

- **Return Value**

0 This function is executed successfully and the socket is not a member.
1 This function is executed successfully and the socket is a member.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.19. qapi_getpeername

Returns the address of the peer connected to the socket in the buffer pointed by the *addr*.

- **Prototype**

```
qapi_Status_t qapi_getpeername ( int32_t handle, struct sockaddr * addr, int * addrlen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

addr:
[Out] Pointer to a user buffer of *sockaraddr* type which is filled by the API with the peer *addr* information.

addrlen:
[In] Specifies the size, in bytes, of the address pointed to by *addr*.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

3.2.20. qapi_getsockname

Returns current address to which the socket is bound in the user provided buffer *addr*.

- **Prototype**

```
qapi_Status_t qapi_getsockname ( int32_t handle, struct sockaddr * addr, int * addrlen );
```

- **Parameters**

handle:
[In] Socket handle returned from *qapi_socket()*.

addr:
[Out] Pointer to a user buffer of *sockaraddr* type which is filled by the API with the peer *addr* information.

addrlen:
[In] Specifies the size, in bytes, of the address pointed to by *addr*.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

A valid handle must be obtained by *qapi_socket()*.

4 Network Security APIs

This chapter describes the QAPIs used for transport layer security (TLS) and datagram transport layer security (DTLS). See ***Appendix A*** for TLS/DTLS supported cipher suites.

TLS and DTLS are used to provide security and data integrity between two peers communicating over TCP or UDP. After a TCP/UDP connection is established, the two peers use a handshake mechanism to establish the keys used for encryption/decryption and data verification. Once the handshake is successful, data can be transmitted/received over the TLS/DTLS connection.

This chapter provides the following APIs:

```
qapi_Net_SSL_Obj_New
qapi_Net_SSL_Con_New
qapi_Net_SSL_Configure
qapi_Net_SSL_Cert_delete
qapi_Net_SSL_Cert_Store
qapi_Net_SSL_Cert_Convert_And_Store
qapi_Net_SSL_Cert_Load
qapi_Net_SSL_Cert_List
qapi_Net_SSL_Fd_Set
qapi_Net_SSL_Accept
qapi_Net_SSL_Connect
qapi_Net_SSL_Shutdown
qapi_Net_SSL_Obj_Free
qapi_Net_SSL_Read
qapi_Net_SSL_Write
```

4.1. Data Types

4.1.1. Enumeration Type

4.1.1.1. enum qapi_Net_SSL_Role_t

SSL object role.

```
typedef enum
{
   QAPI_NET_SSL_SERVER_E = 1,
   QAPI_NET_SSL_CLIENT_E = 2
} qapi_Net_SSL_Role_t;
```

- **Parameters**

Parameter	Description
QAPI_NET_SSL_SERVER_E	Server role.
QAPI_NET_SSL_CLIENT_E	Client role.

4.1.1.2. enum qapi_Net_SSL_Protocol_t

SSL protocol.

```
typedef enum
{
   QAPI_NET_SSL_TLS_E    = 1,
   QAPI_NET_SSL_DTLS_E = 2,
} qapi_Net_SSL_Protocol_t;
```

- **Parameters**

Parameter	Description
QAPI_NET_SSL_TLS_E	TLS protocol.
QAPI_NET_SSL_DTLS_E	DTLS protocol.

4.1.1.3. enum qapi_Net_SSL_Cert_Type_t

SSL certificate type.

```
typedef enum
{
   QAPI_NET_SSL_CERTIFICATE_E = 1,
   QAPI_NET_SSL_CA_LIST_E = 2,
   QAPI_NET_SSL_PSK_TABLE_E = 3,
```

```
    QAPI_NET_SSL_DI_CERT_E = 4
} qapi_Net_SSL_Cert_Type_t;
```

- **Parameters**

Parameter	Description
QAPI_NET_SSL_CERTIFICATE_E	Certificate type.
QAPI_NET_SSL_CA_LIST_E	CA list type.
QAPI_NET_SSL_PSK_TABLE_E	PSK key table type.
QAPI_NET_SSL_DI_CERT_E	Domain Issued Cert type.

4.1.2. Definition Type

4.1.2.1. Maximum Number of Characters in a Certificate or CA List Name.

```
#define QAPI_NET_SSL_MAX_CERT_NAME_LEN (64)
```

4.1.2.2. Maximum Number of Characters in a Domain Name for the Certificates.

```
#define QAPI_NET_SSL_MAX_DOMAIN_NAME_LEN (64)
```

4.1.2.3. Maximum Number of File Names Returned in the qapi_Net_SSL_Cert_List() API.

```
#define QAPI_NET_SSL_MAX_NUM_CERTS (10)
```

4.1.2.4. Maximum Number of Cipher Suites that Can be Configured.

```
#define QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH 8
```

4.1.2.5. Invalid Handle.

```
#define QAPI_NET_SSL_INVALID_HANDLE (0)
```

4.1.2.6. SSL protocol version

```
#define QAPI_NET_SSL_PROTOCOL_UNKNOWN    0x00
#define QAPI_NET_SSL_PROTOCOL_TLS_1_0    0x31
#define QAPI_NET_SSL_PROTOCOL_TLS_1_1    0x32
#define QAPI_NET_SSL_PROTOCOL_TLS_1_2    0x33
#define QAPI_NET_SSL_PROTOCOL_DTLS_1_0   0xEF
#define QAPI_NET_SSL_PROTOCOL_DTLS_1_2   0xED
```

- **Parameters**

Parameter	Description
QAPI_NET_SSL_PROTOCOL_UNKNOWN	Unknown SSL protocol version.
QAPI_NET_SSL_PROTOCOL_TLS_1_0	TLS version 1.0.
QAPI_NET_SSL_PROTOCOL_TLS_1_1	TLS version 1.1.
QAPI_NET_SSL_PROTOCOL_TLS_1_2	TLS version 1.2.
QAPI_NET_SSL_PROTOCOL_DTLS_1_0	DTLS version 1.0.
QAPI_NET_SSL_PROTOCOL_DTLS_1_2	DTLS version 1.2.

4.1.2.7. SSL cipher suites. To be used with qapi_Net_SSL_Configure

```
#define QAPI_NET_TLS_PSK_WITH_RC4_128_SHA                0x008A
#define QAPI_NET_TLS_PSK_WITH_3DES_EDE_CBC_SHA           0x008B
#define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA            0x008C
#define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA            0x008D
#define QAPI_NET_TLS_PSK_WITH_AES_128_GCM_SHA256         0x00A8
#define QAPI_NET_TLS_PSK_WITH_AES_256_GCM_SHA384         0x00A9
#define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA256         0x00AE
#define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA384         0x00AF
#define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA            0x002F
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA        0x0033
#define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA            0x0035
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA        0x0039
#define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA256         0x003C
#define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA256         0x003D
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256     0x0067
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256     0x006B
#define QAPI_NET_TLS_RSA_WITH_AES_128_GCM_SHA256         0x009C
```

```
#define QAPI_NET_TLS_RSA_WITH_AES_256_GCM_SHA384 0x009D
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 0x009E
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 0x009F
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA 0xC004
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA 0xC005
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA 0xC009
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA 0xC00A
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA 0xC00E
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA 0xC00F
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA 0xC013
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA 0xC014
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 0xC023
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 0xC024
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 0xC025
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 0xC026
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 0xC027
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 0xC028
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 0xC029
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 0xC02A
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 0xC02B
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 0xC02C
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 0xC02D
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 0xC02E
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 0xC02F
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 0xC030
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 0xC031
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 0xC032
#define QAPI_NET_TLS_RSA_WITH_AES_128_CCM 0xC09C
#define QAPI_NET_TLS_RSA_WITH_AES_256_CCM 0xC09D
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM 0xC09E
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM 0xC09F
#define QAPI_NET_TLS_RSA_WITH_AES_128_CCM_8 0xC0A0
#define QAPI_NET_TLS_RSA_WITH_AES_256_CCM_8 0xC0A1
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM_8 0xC0A2
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM_8 0xC0A3
#define QAPI_NET_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC13
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 0xCC14
#define QAPI_NET_TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC15
```

- **Parameters**

Parameter	Description
QAPI_NET_TLS_PSK_WITH_RC4_128_SHA	TLS PSK with RC4 128 SHA.

QAPI_NET_TLS_PSK_WITH_3DES_EDE_CBC_SHA	TLS PSK with 3DES EDE CBC SHA.
QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA	TLS PSK with AES 128 CBC SHA.
QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA	TLS PSK with AES 256 CBC SHA.
QAPI_NET_TLS_PSK_WITH_AES_128_GCM_SHA256	TLS PSK with AES_128 GCM SHA256.
QAPI_NET_TLS_PSK_WITH_AES_256_GCM_SHA384	TLS PSK with AES 256 GCM SHA384.
QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA256	TLS PSK with AES 128 CBC SHA256.
QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA384	TLS PSK with AES 256 CBC SHA384.
QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA	Cipher TLS RSA with AES 128 CBC SHA.
QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA	Cipher TLS DHE RSA with AES 128 CBC SHA.
QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA	Cipher TLS RSA with AES 256 CBC SHA.
QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA	Cipher TLS DHE RSA with AES 256 CBC SHA.
QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA256	Cipher TLS RSA with AES 128 CBC SHA256.
QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA256	Cipher TLS RSA with AES 256 CBC SHA256.
QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	Cipher TLS DHE RSA with AES 128 CBC SHA256.
QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	Cipher TLS DHE RSA with AES 256 CBC SHA256.
QAPI_NET_TLS_RSA_WITH_AES_128_GCM_SHA256	Cipher TLS RSA with AES 128 GCM SHA256.
QAPI_NET_TLS_RSA_WITH_AES_256_GCM_SHA384	Cipher TLS RSA with AES 256 GCM SHA384.
QAPI_NET_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	Cipher TLS DHE RSA with AES 128 GCM

	SHA256.
QAPI_NET_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	Cipher TLS DHE RSA with AES 256 GCM SHA384.
QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	Cipher TLS ECDH ECDSA with AES 128 CBC SHA.
QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	Cipher TLS ECDH ECDSA with AES 256 CBC SHA.
QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	Cipher TLS ECDHE ECDSA with AES 128 CBC SHA.
QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	Cipher TLS ECDHE ECDSA with AES 256 CBC SHA.
QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	Cipher TLS ECDH RSA with AES 128 CBC SHA.
QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	Cipher TLS ECDH RSA with AES 256 CBC SHA.
QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Cipher TLS ECDHE RSA with AES 128 CBC SHA.
QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Cipher TLS ECDHE RSA with AES 256 CBC SHA.
QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	Cipher TLS ECDHE ECDSA with AES 128 CBC SHA256.
QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	Cipher TLS ECDHE ECDSA with AES 256 CBC SHA384.
QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	Cipher TLS ECDH ECDSA with AES 128 CBC SHA256.
QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	Cipher TLS ECDH ECDSA with AES 256 CBC SHA384.
QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Cipher TLS ECDHE RSA with AES 128 CBC

	SHA256.
QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Cipher TLS ECDHE RSA with AES 256 CBC SHA384.
QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	Cipher TLS ECDHE RSA with AES 128 CBC SHA256.
QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	Cipher TLS ECDHE RSA with AES 256 CBC SHA384.
QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Cipher TLS ECDH RSA with AES 128 CBC SHA256.
QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Cipher TLS ECDH RSA with AES 256 CBC SHA384.
QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	Cipher TLS ECDHE ECDSA with AES 128 GCM SHA256.
QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	Cipher TLS ECDHE ECDSA with AES 256 GCM SHA384.
QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Cipher TLS ECDH ECDSA with AES 128 GCM SHA256.
QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Cipher TLS ECDH ECDSA with AES 256 GCM SHA384.
QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	Cipher TLS ECDHE RSA with AES 128 GCM SHA256.
QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	Cipher TLS ECDHE RSA with AES 256 GCM SHA384.
QAPI_NET_TLS_RSA_WITH_AES_128_CCM	Cipher TLS ECDH RSA with AES 128 GCM SHA256.
QAPI_NET_TLS_RSA_WITH_AES_256_CCM	Cipher TLS ECDH RSA with AES 256 GCM SHA384.
QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM	Cipher TLS RSA with AES 128 CCM.

QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM	Cipher TLS RSA with AES 256 CCM.
QAPI_NET_TLS_RSA_WITH_AES_128_CCM_8	Cipher TLS DHE RSA with AES 128 CCM.
QAPI_NET_TLS_RSA_WITH_AES_256_CCM_8	Cipher TLS DHE RSA with AES 256 CCM.
QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM_8	Cipher TLS RSA with AES 128 CCM 8.
QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM_8	Cipher TLS RSA with AES 256 CCM 8.
QAPI_NET_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	Cipher TLS ECDHE RSA with CHACHA20 POLY1305 SHA256.
QAPI_NET_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	Cipher TLS ECDHE ECDSA with CHACHA20 POLY1305 SHA256.
QAPI_NET_TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	Cipher TLS DHE RSA with CHACHA20 POLY1305 SHA256.

4.1.2.8. Maximum Certificate Authority List Entries Allowed for Conversion to Binary Format.

```
#define QAPI_NET_SSL_MAX_CA_LIST 10
```

4.1.3. Tpyedefs

4.1.3.1. typedef uint32_t qapi_Net_SSL_Obj_Hdl_t

Handle to an SSL object.

This is obtained from a call to *qapi_Net_SSL_Obj_New()*. The handle is freed with a call to *qapi_Net_SSL_Obj_Free()*.

4.1.3.2. typedef uint32_t qapi_Net_SSL_Con_Hdl_t

Handle to an SSL connection.

This is obtained from a call to *qapi_Net_SSL_Con_New().* The handle is freed with a call to *qapi_Net_SSL_Shutdown()*.

4.1.3.3. typedef const void* qapi_Net_SSL_Cert_t

Internal certificate format. The certificate is in a binary format optimized for speed and size.

4.1.3.4. typedef const void* qapi_Net_SSL_CAList_t

Internal CA list format. The CA list is in a binary format optimized for speed and size.

4.1.3.5. typedef const void* qapi_Net_SSL_PSKTable_t

Internal *psk_table* format. PSK table is in an optimized binary format.

4.1.4. Structure Type

4.1.4.1. Struct qapi_Net_SSL_Verify_Policy_t

Structure to specify the certificate verification policy.

```
typedef struct __qapi_Net_SSL_Verify_Policy_s
{
    uint8_t domain;
    uint8_t time_Validity;
    uint8_t send_Alert;
    char match_Name[QAPI_NET_SSL_MAX_CERT_NAME_LEN];
} qapi_Net_SSL_Verify_Policy_t;
```

- **Parameters**

Type	Parameter	Description
uint8_t	*domain*	TRUE to verify certificate commonName against the peer's domain name.
uint8_t	*time_Validity*	TRUE to verify certificate time validity.
uint8_t	*send_Alert*	TRUE to immediately send a fatal alert on detection of an untrusted certificate.
char	*match_Name*	Name to match against the common name or altDNSNames of the certificate. See *QAPI_NET_SSL_MAX_CERT_NAME_LEN*.

4.1.4.2. Struct qapi_Net_SSL_Identifier_t

Structure to get the Identifier from the certificate.

```
typedef struct __qapi_Net_SSL_Identifier_s
{
    qapi_Net_SSL_Identifier_Type_t identifier_Type;
    char identifier_Name [QAPI_NET_SSL_MAX_DOMAIN_NAME_LEN];
} qapi_Net_SSL_Identifier_t;
```

- **Parameters**

Type	Parameter	Description
qapi_Net_SSL_Identifier_Type_t	*identifier_Type*	Type of Identifier need to extract from certificate.
char	*identifier_Name*	Name to altDNSNames or altURIs or Common Name of the certificate. See *QAPI_NET_SSL_MAX_CERT_NAME_LEN*.

4.1.4.3. Struct qapi_Net_SSL_Config_t

Structure to configure an SSL connection.

```
typedef struct __qapi_Net_SSL_Config_s
{
    uint16_t protocol;
    uint16_t cipher [QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH];
    qapi_Net_SSL_Verify_Policy_t verify;
    uint16_t max_Frag_Len;
    uint16_t max_Frag_Len_Neg_Disable;
    uint16_t sni_Name_Size;
    char *sni_Name;
} qapi_Net_SSL_Config_t;
```

- **Parameters**

Type	Parameter	Description
uint16_t	*protocol*	Protocol to use. See *QAPI_NET_SSL_PROTOCOL_∗*.
uint16_t	*cipher*	Cipher to use. See SSL cipher suites *QAPI_NET_TLS∗* and

		QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH.
qapi_Net_SSL_Verify_Policy_t	*verify*	Certificate verification policy.
uint16_t	*max_Frag_Len*	Maximum fragment length in bytes.
uint16_t	*max_Frag_Len-_Neg_Disable*	Whether maximum fragment length negotiation is allowed. See RFC 6066.
uint16_t	*sni_Name_Size*	Length of the SNI server name.
char *	*sni_Name*	Server name for SNI.

4.1.4.4. Struct qapi_Net_SSL_Cert_List_t

Structure to get a list of certificates stored in nonvolatile memory.

```
typedef struct __qapi_Net_SSL_Cert_List_s
{
    char name[QAPI_NET_SSL_MAX_NUM_CERTS][QAPI_NET_SSL_MAX_CERT_NAME_LEN];
} qapi_Net_SSL_Cert_List_t;
```

- **Parameters**

Type	Parameter	Description
char	*name*	Certificate name. See *QAPI_NET_SSL_MAX_NUM_CERTS* and *QAPI_NET_SSL_MAX_CERT_NAME_LEN*.

4.1.4.5. Struct qapi_Net_SSL_CERT_t

SSL client certificate information for conversion and storage.

```
typedef struct __qapi_Net_SSL_CERT_s
{
    uint8_t *cert_Buf;
    uint32_t cert_Size;
    uint8_t *key_Buf;
    uint32_t key_Size;
    uint8_t *pass_Key;
} qapi_Net_SSL_CERT_t;
```

- **Parameters**

Type	Parameter	Description
uint8_t *	*cert_Buf*	Client certificate buffer.
uint32_t	*cert_Size*	Client certificate buffer size.
uint8_t *	*key_Buf*	Private key buffer.
uint32_t	*key_Size*	Private key buffer size.
uint8_t *	*pass_Key*	Password phrase.

4.1.4.6. Struct qapi_NET_SSL_CA_Info_t

SSL certificate authority list information.

```
typedef struct __qapi_NET_SSL_CA_Info_s
{
    uint8_t *ca_Buf;
    uint32_t ca_Size;
} qapi_NET_SSL_CA_Info_t;
```

- **Parameters**

Type	Parameter	Description
uint8_t *	*ca_Buf*	Certificate authority list buffer.
uint32_t	*ca_Size*	Certificate authority list buffer size.

4.1.4.7. Struct qapi_Net_SSL_CA_List_t

SSL certificate authority information for conversion and storage.

```
typedef struct __qapi_Net_SSL_CA_List_s
{
    uint32_t ca_Cnt;
    qapi_NET_SSL_CA_Info_t *ca_Info[QAPI_NET_SSL_MAX_CA_LIST];
} qapi_Net_SSL_CA_List_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*ca_Cnt*	Certificate authority list count.
qapi_NET_SSL_CA_Info_t *	*ca_Info*	Certificate authority list info.

4.1.4.8. Struct qapi_Net_SSL_PSK_Table_t

SSL PSK table information for conversion and storage.

```
typedef struct __qapi_Net_SSL_PSK_Table_s
{
    uint32_t psk_Size;
    uint8_t *psk_Buf;
} qapi_Net_SSL_PSK_Table_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*psk_Size*	PSK table buffer size.
uint8_t *	*psk_Buf*	PSK table buffer.

4.1.4.9. Struct qapi_Net_SSL_Cert_Info_t

SSL general certification information for conversion and storage for client certificates, CA lists, and PSK tables.

```
typedef struct __qapi_Net_SSL_Cert_Info_s
{
    qapi_Net_SSL_Cert_Type_t cert_Type;
    union
    {
        qapi_Net_SSL_CERT_t cert;
        qapi_Net_SSL_CA_List_t ca_List;
        qapi_Net_SSL_PSK_Table_t psk_Tbl;
        qapi_Net_SSL_DI_Cert_t di_cert;
    } info;
} qapi_Net_SSL_Cert_Info_t;
```

- **Parameters**

Type	Parameter	Description
qapi_Net_SSL_Cert_Type_t	*cert_Type*	Certification type.
union __qapi_Net_SSL_Cert_Info_s	*info*	Certificate information.

4.1.4.10. union __qapi_Net_SSL_Cert_Info_s.info

Certification information.

```
union
{
    qapi_Net_SSL_CERT_t cert;
    qapi_Net_SSL_CA_List_t ca_List;
    qapi_Net_SSL_PSK_Table_t psk_Tbl;
    qapi_Net_SSL_DI_Cert_t di_cert;
} info;
```

- **Parameters**

Type	Parameter	Description
qapi_Net_SSL_CERT_t	*cert*	Certificate.
qapi_Net_SSL_CA_List_t	*ca_List*	CA list.
qapi_Net_SSL_PSK_Table_t	*psk_Tbl*	PSK table.
qapi_Net_SSL_DI_Cert_t	*di_cert*	Domain issued certificate.

4.1.4.11. Struct qapi_Net_SSL_DI_Cert_t

SSL Domain Issued Cert info for conversion and storage.

```
typedef struct __qapi_Net_SSL_DI_Cert_s
{
    uint32_t di_Cert_Size;
    uint8_t *di_Cert_Buf;
} qapi_Net_SSL_DI_Cert_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*di_Cert_Size*	Domain issued cert buffer size.
uint8_t *	*di_Cert_Buf*	Domain issued cert buffer.

4.2. API Functions

4.2.1. qapi_Net_SSL_Obj_New

Creates a new SSL object (server or client).

- **Prototype**

```
qapi_Net_SSL_Obj_Hdl_t qapi_Net_SSL_Obj_New ( qapi_Net_SSL_Role_t role );
```

- **Parameters**

role:
[In] Server or client role.

- **Return Value**

SSL object handle on success.
QAPI_NET_SSL_HDL_NULL on error (out of memory).

- **Dependencies**

This function must be called before using any other SSL function.

4.2.2. qapi_Net_SSL_Con_New

Creates an SSL connection handle for an SSL object.

- **Prototype**

```
qapi_Net_SSL_Con_Hdl_t qapi_Net_SSL_Con_New ( qapi_Net_SSL_Obj_Hdl_t hdl,qapi_Net_SSL_Protocol_t prot );
```

- **Parameters**

hdl:
[In] SSL object handle.

prot:
[In] Protocol to be used for this connection.

- **Return Value**

SSL connection handle on success.
QAPI_NET_SSL_HDL_NULL on error (out of memory).

- **Dependencies**

A valid handle must be obtained by *qapi_Net_SSL_Obj_New*().

4.2.3. qapi_Net_SSL_Configure

Configures an SSL connection regarding protocol and cipher, certificate validation criteria, maximum fragment length, and disable fragment length negotiation.

The SSL protocol and up to 8 ciphers can be configured in the client context. The *SSL_VERIFY_POLICY* verify structure (and matchName) specify how the SSL certificate will be verified during the SSL handshake:

- If *verify.domain*=1, the certificate domain name will be checked against matchName.
- If *verify.timeValidity*=1, the certificate will be checked for expiration.
- The certificate itself is always checked against the CAList. If a CAList is not present in the SSL context, the certificate is implicitly trusted.
- If *verify.sendAlert*=1, an SSL alert is sent if the certificate fails any of the tests. An error is also returned to the application, which subsequently closes the connection. If *verify.sendAlert*=0, an error is returned by *SSL_connect()*, and it is up to the application to decide what to do.

In SSL, a smaller fragment length helps in efficient memory utilization and to minimize latency. In Client mode, a maximum fragment length of 1 KB is negotiated during handshake using TLS extensions. If the peer server does not support the extension, the default maximum size of 16 KB is used.

SSL_configure provides two fields, *max_frag_len* and *max_frag_len_neg_disable*, to override the above behavior. *max_frag_len_neg_disable* applies only in Client mode.

If negotiation is allowed (i.e, *max_frag_len_neg_disable*=0), *max_frag_len* must be set to one of these four values, according to RFC 6066:

1 512
2 1024
3 2048
4 4096 Other values are not permitted.

max_frag_len is applicable in Client or Server mode. Server mode does not support a maximum fragment length TLS extension.

There can be scenarios where the peer does not support the maximum fragment length TLS extension, but the maximum fragment length is inferred. In that case, the user may choose to configure *max_frag_len* andset *max_frag_len_neg_disable* to 1 to disable negotiation and still get the benefits of a smaller fragment length. When negotiation is disabled, any value<16 KB can be configured for *max_frag_len*. Then the above limitations do not apply.

An error is returned and the connection is closed if any incoming record exceeds *max_frag_len*.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Configure ( qapi_Net_SSL_Con_Hdl_t ssl, qapi_Net_SSL_Config_t * cfg );
```

- **Parameters**

ssl:
[In] Connection handle.

cfg:
[In] Configuration parameters.

- **Return Value**

QAPI_OK	This function is executed successfully..
QAPI_ERR_INVALID_PARAM_SSL	It fails to execute this function (configuration is invalid).

- **Dependencies**

A valid connection handle must be obtained by *qapi_Net_SSL_Con_New*().

4.2.4. qapi_Net_SSL_Cert_delete

Deletes an encrypted certificate, CA list, or a PSK table from nonvolatile memory.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Cert_delete ( char * name, qapi_Net_SSL_Cert_Type_t type );
```

- **Parameters**

name:
[In] Name of the certificate, CA list, or PSK table. The maximum length of the name allowed is *QAPI_NET_SSL_MAX_CERT_NAME_LE*, including the NULL character.

type:
[In] Type of data (certificate or CA list) to store. Could be either *QAPI_NET_SSL_CERTIFICATE_E*, *QAPI_NET_SSL_CA_LIST_E*, or *QAPI_NET_SSL_PSK_TABLE_E*.

- **Return Value**

0 on success.
Negative value on error.

- **Dependencies**

None.

4.2.5. qapi_Net_SSL_Cert_Store

Stores an internal certificate, CA list, or a PSK table in nonvolatile memory in encrypted form. The certificate, CA list and PSK are both in binary format optimized for speed and size.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Cert_Store ( const char * name, qapi_Net_SSL_Cert_Type_t type, qapi_Net_SSL_Cert_t cert, uint32_t size);
```

- **Parameters**

name:
[In] Name of the certificate, CA list, or PSK table. The maximum length of the name allowed is *QAPI_NET_SSL_MAX_CERT_NAME_LE*, including the NULL character.

type:
[In] Type of data (certificate or CA list) to store. Could be either *QAPI_NET_SSL_CERTIFICATE_E*, *QAPI_NET_SSL_CA_LIST_E*, or *QAPI_NET_SSL_PSK_TABLE_E*.

cert:
[In] Address of the file containing the certificate in SSL internal format (*.bin file).

size:
[In] Size of the certificate file.

- **Return Value**

0 on success.
Negative value on error.

- **Dependencies**

None.

4.2.6. qapi_Net_SSL_Cert_Convert_And_Store

Converts certificates, CA lists from .PEM, .DER, or .P7B, and PSK tables to binary format and stores them in nonvolatile memory in encrypted form. The certificate is in binary format optimized for speed and size. Only one of these types can be converted and stored at a time.

The maximum number of CA lists that are supported for conversion and storage in binary format is *QAPI_NET_SSL_MAX_CA_LIST*.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Cert_Convert_And_Store ( qapi_Net_SSL_Cert_Info_t * cert_info, const uint8_t * cert_name );
```

- **Parameters**

cert_info:
[In] Information pertaining to either the client certificate, CA lists in .PEM, .DER, or .P7B format, or PSK tables.

cert_name:
[In] Name of the certificate, CA list, or PSK table that the *cert_info* is to be stored under after the conversion.

- **Return Value**

0 on success.
Negative value on error.

- **Dependencies**

None.

4.2.7. qapi_Net_SSL_Cert_Load

Reads an encrypted certificate, CA list, or PSK table from nonvolatile memory, decrypts it, and then adds it to the SSL object.

- Certificate – Loads a client or server certificate to the SSL object. In the server SSL, the context is required to have at least one certicfiate, but multiple may be added.

- Certificate Authority (CA) list – Enables the SSL object to perform certificate validation on the peer's certificate. Only one CA list can be set, thus the CA list must include all root certificates required for the Session

- PSK table – Holds a list of preshared keys (PSK) to load SSL context. Only one PSK table can be set, thus the PSK table must include all PSK entries required for the session.

Certificates, CA lists, or a PSK table must be added before the *qapi_Net_SSL_Connect()* or *qapi_Net_SSL_Accept()* APIs are called.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Cert_Load ( qapi_Net_SSL_Obj_Hdl_t hdl, qapi_Net_SSL_Cert_Type_t type, const char * name );
```

- **Parameters**

hdl:
[In] SSL object handle.

type:
[In] Type of data (certificate or CA list) to load. Could be either *QAPI_NET_SSL_CERTIFICATE_E*, *QAPI_NET_SSL_CA_LIST_E*, or *QAPI_NET_SSL_PSK_TABLE_E*.

- **Return Value**

0 on success.
Negative value on error.

- **Dependencies**

A valid handle must be obtained by *qapi_Net_SSL_Obj_New()*.

4.2.8. qapi_Net_SSL_Cert_List

Gets a list of encrypted certificates, CA lists, or a PSK tables stored in nonvolatile memory. The structure *__qapi_Net_SSL_Cert_List_s* must be allocated by the caller.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Cert_List ( qapi_Net_SSL_Cert_Type_t type, qapi_Net_SSL_Cert_List_t * list );
```

- **Parameters**

type:
[In] Type of data (certificate or CA list) to store. This can be either *QAPI_NET_SSL_CERTIFICATE_E*, *QAPI_NET_SSL_CA_LIST_E*, or *QAPI_NET_SSL_PSK_TABLE_E*.

list:
[Out] List of file names.

- **Return Value**

Number of files on success.
0 on error.

- **Dependencies**

None.

4.2.9. qapi_Net_SSL_Fd_Set

Attaches a given socket descriptor to the SSL connection. The SSL connection inherits the behavior of the socket descriptor (zero-copy/nonzero-copy, blocking/nonblocking, etc.).

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Fd_Set ( qapi_Net_SSL_Con_Hdl_t ssl, uint32_t fd );
```

- **Parameters**

ssl:
[In] SSL connection handle.

fd:
[In] FD socket descriptor.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERR_INVALID_PARAM_SSL	It fails to execute this function.

- **Dependencies**

A valid SSL connection handle must be obtained by *qapi_Net_SSL_Con_New()*, and a valid socket handle must be created firstly.

4.2.10. qapi_Net_SSL_Accept

Accepts an incoming SSL connection from the client.

This should be called only by a server SSL object. This will respond to the incoming client Hello message and complete the SSL handshake.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Accept ( qapi_Net_SSL_Con_Hdl_t ssl );
```

- **Parameters**

ssl:
[In] SSL connection handle.

- **Return Value**

QAPI_SSL_OK_HS	This function is executed successfully.
*QAPI_ERR_**:	It fails to execute this function.

- **Dependencies**

A valid server SSL connection handle must be obtained by *qapi_Net_SSL_Con_New()*.

4.2.11. qapi_Net_SSL_Connect

Initiates an SSL handshake. Called only by a client SSL object.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Connect ( qapi_Net_SSL_Con_Hdl_t ssl );
```

- **Parameters**

ssl:
[In] SSL connection handle.

- **Return Value**

QAPI_SSL_OK_HS	This function is executed successfully.
*QAPI_ERR_**	It fails to execute this function.

- **Dependencies**

A valid SSL connection handle must be obtained by *qapi_Net_SSL_Con_New()*.

4.2.12. qapi_Net_SSL_Shutdown

Closes an SSL connection.

The connection handle will be freed in this API. The socket must be closed explicitly after this call. See *qapi_socketclose()*.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Shutdown ( qapi_Net_SSL_Con_Hdl_t ssl );
```

- **Parameters**

ssl:
[In] SSL connection handle.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERR_INVALID_PARAM_SSL	It fails to execute this function (invalid connection handle).

- **Dependencies**

A valid SSL connection handle must be obtained by *qapi_Net_SSL_Con_New()*.

4.2.13. qapi_Net_SSL_Obj_Free

Frees the SSL object handle.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Obj_Free ( qapi_Net_SSL_Obj_Hdl_t hdl );
```

- **Parameters**

hdl:
[In] SSL object handle.

- **Return Value**

QAPI_OK on success.
Other values on error.

- **Dependencies**

A valid SSL object handle must be obtained by *qapi_Net_SSL_Obj_New()*. All connections belonging to

this handle must be closed before calling this API.

4.2.14. qapi_Net_SSL_Read

Reads data received over the SSL connection.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Read ( qapi_Net_SSL_Con_Hdl_t hdl, void * buf, uint32_t size );
```

- **Parameters**

hdl:
[In] SSL coonection handle.

buf:
[Out] Buffer to hold received data. Must be allocated by the application.

size:
[In] Size of the buffer in bytes.

- **Return Value**

The number of bytes available in the buffer.
*QAPI_ERR_** on error.

- **Dependencies**

The SSL handshake must be completed successfully before calling this API. Depending on the underlying socket associated with the SSL connection, the API will be blocking/nonblocking, etc. The select API can be used to check if there is any data available.

4.2.15. qapi_Net_SSL_Write

Sends data over the SSL connection.

- **Prototype**

```
qapi_Status_t qapi_Net_SSL_Write ( qapi_Net_SSL_Con_Hdl_t hdl, void * buf, uint32_t size );
```

- **Parameters**

hdl:
[In] SSL connection handle.

buf:
[In] Buffer with the data to be sent.

size:
[In] Size of the buffer in bytes.

- **Return Value**

The number of bytes sent.
*QAPI_ERR_** on error.

- **Dependencies**

The SSL handshake must be completed successfully before calling this API. Depending on the underlying socket associated with the SSL connection, the API will be blocking/nonblocking, etc.

5 Network Services APIs

This chapter provides the following network services and utilities APIs:

```
qapi_Net_Get_All_Ifnames
inet_pton
inet_ntop
qapi_Net_Interface_Get_Physical_Address
qapi_Net_Interface_Exist
qapi_Net_IPv4_Config
qapi_Net_Ping
qapi_Net_Ping_2
qapi_Net_Ping_3
qapi_Net_IPv4_Route
qapi_Net_Ping6
qapi_Net_Ping6_2
qapi_Net_Ping6_3
qapi_Net_IPv6_Get_Address
qapi_Net_IPv6_Route
qapi_Net_IPv6_Get_Scope_ID
```

5.1. Data Types

5.1.1. Enumeration Type

5.1.1.1. enum qapi_Net_Route_Command_t

Commands for routing QAPI net services.

```
typedef enum
{
    QAPI_NET_ROUTE_ADD_E,
    QAPI_NET_ROUTE_DEL_E,
    QAPI_NET_ROUTE_SHOW_E,
    QAPI_NET_ROUTE_MAX_E
} qapi_Net_Route_Command_t;
```

- **Parameters**

Parameter	Description
QAPI_NET_ROUTE_ADD_E	Add route.
QAPI_NET_ROUTE_DEL_E	Delete route.
QAPI_NET_ROUTE_SHOW_E	Show routes.

5.1.1.2. enum qapi_Net_IPv4cfg_Command_t

Commands for the IPv4 configuration QAPI.

```
typedef enum
{
    QAPI_NET_IPV4CFG_QUERY_E,
    QAPI_NET_IPV4CFG_STATIC_IP_E,
    QAPI_NET_IPV4CFG_DHCP_IP_E,
    QAPI_NET_IPV4CFG_AUTO_IP_E,
    QAPI_NET_IPV4CFG_MAX_E
} qapi_Net_IPv4cfg_Command_t;
```

- **Parameters**

Parameter	Description
QAPI_NET_IPV4CFG_QUERY_E	Get the IPv4 parameters of an interface, such as IP address, subnet mask, and default gateway.
QAPI_NET_IPV4CFG_STATIC_IP_E	Assign the IPv4 address, subnet mask, and default gateway.
QAPI_NET_IPV4CFG_DHCP_IP_E	Run the DHCPv4 client to obtain IPv4 parameters from the DHCPv4 server.
QAPI_NET_IPV4CFG_AUTO_IP_E	Run auto-IP (automatic private IP addressing).

5.1.2. Definition Type

5.1.2.1. Verifies Whether the IPv4 Address is Multicast

This macro returns 1 if the passed IPv4 address is multicast. IPv4 multicast addresses are in the range 224.0.0.0 through 239.255.255.255.

```
#define   QAPI_IPV4_IS_MULTICAST(ipv4_Address) \
      (((long)(ipv4_Address) &   0xf0000000) == 0xe0000000)
```

- **Parameters**

Parameter	Description
ipv4_Address	IPv4 address to check; must be in host order.

- **Return Value**

1 if the IPv4 address is multicast, 0 otherwise.

5.1.2.2. Default Maximum Length for Interface Names

```
#define IF_NAMELEN     20
```

5.1.2.3. Maximum IPv4 Routing Configurations

```
#define QAPI_NET_IPV4_MAX_ROUTES      (3)
```

5.1.2.4. Checks Whether the IPv6 Address is Link Local

This macro returns 1 if the passed IPv6 address is link local. The link local address format is fe80::/64. The first 10 bits of the address are 1111111010, followed by 54 zeros, followed by 64 bits of the interface identifier.

- **Parameters**

Parameter	Description
ipv4_Address	IPv6 address to check.

- **Return Value**

1 if the IPv6 address is link local, 0 otherwise.

5.1.2.5. Checks Whether the IPv6 Address is Multicast

```
#define QAPI_IS_IPV6_MULTICAST(ipv6_Address) \
```

```
    (ipv6_Address[0] == 0xff)
```

- **Parameters**

Parameter	Description
Ipv6_Address	Ipv6 address to check.

- **Return Value**

1 if the Ipv6 address is multicast, 0 otherwise.

5.1.2.6. Maximum IPv6 Routing Configurations

```
#define QAPI_NET_IPV6_MAX_ROUTES       (3)
```

5.1.2.7. Maximum Length for the Interface Name

```
#define QAPI_NET_IFNAME_LEN        12
```

5.1.2.8. IPV4 Ping Bitmask

```
#define QAPI_NET_PING_V4_DST_ADDR_MASK    0x0001
#define QAPI_NET_PING_V4_SRC_ADDR_MASK    0x0002
#define QAPI_NET_PING_V4_PKT_SIZE_MASK    0x0004
#define QAPI_NET_PING_V4_TIMEOUT_MASK     0x0008
#define QAPI_NET_PING_V4_TTL_MASK         0x0010
```

5.1.2.9. IPV6 Ping Bitmask

```
#define QAPI_NET_PING_V6_DST_ADDR_MASK    0x0001
#define QAPI_NET_PING_V6_SRC_ADDR_MASK    0x0002
#define QAPI_NET_PING_V6_PKT_SIZE_MASK    0x0004
#define QAPI_NET_PING_V6_TIMEOUT_MASK     0x0008
#define QAPI_NET_PING_V6_IF_NAME_MASK     0x0010
#define QAPI_NET_PING_V6_TTL_MASK         0x0020
```

5.1.3. Structure Type

5.1.3.1. struct qapi_Net_Ping_V4_t

IPv4 ping input.

```
typedef struct qapi_Net_Ping_V4_s
{
    uint32_t ipv4_addr;
    uint32_t ipv4_src;
    uint32_t size;
    uint32_t timeout;
} qapi_Net_Ping_V4_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*ipv4_addr*	Destination to ping.
uint32_t	*ipv4_src*	Source address.
uint32_t	*size*	Packet size.
uint32_t	*timeout*	Timeout value (in ms).

5.1.3.2. qapi_Net_Ping_V4_R2_t

IPv4 ping input.

```
typedef struct qapi_Net_Ping_V4_R2_s
{
        uint32_t bitmask;
        uint32_t ipv4_addr;
        uint32_t ipv4_src;
        uint32_t size;
        uint32_t timeout;
        uint32_t ttl;
} qapi_Net_Ping_V4_R2_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*bitmask*	Bitmask.
uint32_t	*ipv4_addr*	Destination to ping.
uint32_t	*ipv4_src*	Source address.
uint32_t	*size*	Packet size.
uint32_t	*timeout*	Timeout value (in ms).
uint32_t	*ttl*	Time to live (TTL) or hop limit is a mechanism that limits the lifespan or lifetime of data in a computer or network.

5.1.3.3. struct qapi_Net_IPv4_Route_t

IPv4 routing object.

```
typedef struct
{
    uint32_t RSVD;
    uint32_t ipRouteDest;
    uint32_t ipRouteMask;
    uint32_t ipRouteNextHop;
    uint32_t ipRouteIfIndex;
    uint32_t ipRouteProto;
    char     ifName[IF_NAMELEN];
} qapi_Net_IPv4_Route_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*RSVD*	Reserved.
uint32_t	*ipRouteDest*	Destination IPv4 address of this route.
uint32_t	*ipRouteMask*	Indicates the mask to be logically ANDed with the destination address before being compared to the value in the ipRouteDest field.
uint32_t	*ipRouteNextHop*	IPv4 address of the next hop of this route.
uint32_t	*ipRouteIfIndex*	Index value that uniquely identifies the local interface through which the next hop of this route should be reached.

uint32_t	*ipRouteProto*	Routing mechanism via which this route was learned.
char	*ifName*	Textual name of the interface.

5.1.3.4. struct qapi_Net_IPv4_Route_List_t

IPv4 routing objects list.

```
typedef struct
{
    uint32_t                route_Count;
    qapi_Net_IPv4_Route_t   route[QAPI_NET_IPV4_MAX_ROUTES];
} qapi_Net_IPv4_Route_List_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*route_Count*	Number of *qapi_Net_IPv4_Route_t* arrays in the routing table.
qapi_Net_IPv4_Route_t	*route*	Array of *qapi_Net_IPv4_Route_t* types.

5.1.3.5. struct qapi_Net_Ping_V6_t

IPv6 ping input.

```
typedef struct qapi_Net_Ping_V6_s
{
    uint8_t ipv6_addr[16];
    uint8_t ipv6_src[16];
    uint32_t size;
    uint32_t timeout;
    char    *ifname;
} qapi_Net_Ping_V6_t;
```

- **Parameters**

Type	Parameter	Description
uint8_t	*ipv6_addr*	Destination to ping.

uint8_t	*ipv6_src*	Source address.
uint32_t	*size*	Packet size.
uint32_t	*timeout*	Timeout value (in ms).
char*	*ifname*	Interface name.

5.1.3.6. qapi_Net_Ping_V6_R2_t

IPv6 ping input.

```
typedef struct qapi_Net_Ping_V6_R2_s
{
    uint32_t bitmask;
    uint32_t ipv6_addr[16];
    uint32_t ipv6_src[16];
    uint32_t size;
    uint32_t timeout;
    char    *ifname;
    uint32_t ttl;
} qapi_Net_Ping_V6_R2_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*bitmask*	Bitmask.
uint8_t	*ipv6_addr*	Destination to ping.
uint8_t	*ipv6_src*	Source address.
uint32_t	*size*	Packet size.
uint32_t	*timeout*	Timeout value (in ms).
char*	*ifname*	Interface name.
uint32_t	*ttl*	Time to live (TTL) or hop limit is a mechanism that limits the lifespan or lifetime of data in a computer or network.

5.1.3.7. struct qapi_Net_IPv6_Route_t

IPv6 routing object.

```
typedef struct
{
    uint8_t    ipv6RouteDest[16];
    uint32_t ipv6RoutePfxLength;
    uint8_t    ipv6RouteNextHop[16];
    uint32_t ipv6RouteProtocol;
    uint32_t ipv6RouteIfIndex;
    char         ifName[IF_NAMELEN];
} qapi_Net_IPv6_Route_t;
```

- **Parameters**

Type	Parameter	Description
uint8_t	*ipv6RouteDest*	Destination IPv6 address of this route.
uint32_t	*ipv6RoutePfxLength*	Indicates the prefix length of the destination address.
uint8_t	*ipv6RouteNextHop*	Address of the next system en route.
uint32_t	*ipv6RouteProtocol*	Routing mechanism via which this route was learned.
uint32_t	*ipv6RouteIfIndex*	Index value that uniquely identifies the local interface through which the next hop of this route should be reached.
char	*ifName*	Textual name of the interface.

5.1.3.8. struct qapi_Net_IPv6_Route_List_t

IPv6 routing objects list.

```
typedef struct
{
    uint32_t                    route_Count;
    qapi_Net_IPv6_Route_t    route[QAPI_NET_IPV6_MAX_ROUTES];
} qapi_Net_IPv6_Route_List_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*route_Count*	Number of *qapi_Net_IPv6_Route_t* arrays in the routing table.
qapi_Net_IPv6_Route_t	*route*	Array of type qapi_Net_IPv6_Route_t.

5.1.3.9. struct qapi_Net_Ifnameindex_t

Network interface object.

```
typedef struct
{
    uint32_t if_Index;
    char     interface_Name[QAPI_NET_IFNAME_LEN];
    qbool_t  if_Is_Up;
} qapi_Net_Ifnameindex_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*if_Index*	if_Index in RFC 1213-mib2, which ranges from 1 to the returned value of *qapi_Net_Get_Number_of_Interfaces()* if the value is >= 1.
char	*interface_Name*	Interface name (NULL terminated).
qbool_t	*if_Is_Up*	TRUE if the interface is up, FALSE if interface is not up (e.g., down or testing).

5.1.3.10. struct qapi_Ping_Info_Resp_t

Ping response structure.

```
typedef struct qapi_Ping_Info_Resp_s
{
   int ptype;
   int pcode;
   char perror[128];
} qapi_Ping_Info_Resp_t;
```

- **Parameters**

Type	Parameter	Description
int	*ptype*	ICMP type for the ping.
int	*pcode*	ICMP code for the ping.
char	*perror*	Response description for the ping.

5.1.3.11. struct qapi_Ping_Info_Resp_R2_t

Ping response structure.

```
typedef struct qapi_Ping_Info_Resp_R2_s
{
    int ptype;
    int pcode;
    char perror[128];
    uint8_t ttl;
} qapi_Ping_Info_Resp_R2_t;
```

- **Parameters**

Type	Parameter	Description
int	*ptype*	ICMP type for the ping.
int	*pcode*	ICMP code for the ping.
char	*perror*	Response description for the ping.
uint8_t	*ttl*	Time to live (TTL) or hop limit is a mechanism that limits the lifespan or lifetime of data in a computer or network.

5.2. API Functions

5.2.1. qapi_Net_Get_All_Ifnames

Retrieves the textual names of all network interfaces.

- **Prototype**

```
int32_t qapi_Net_Get_All_Ifnames (qapi_Net_Ifnameindex_t *if_Name_Index);
```

- **Parameters**

f_Name_Index:
[Out] Array to contain the retrieved information.

- **Return Value**

Number of network interfaces

- **Dependencies**

None.

5.2.2. inet_pton

Parses the passed address string into an IPv4/IPv6 address.

- **Prototype**

```
int32_t inet_pton(int32_t af, const char *src, void *dst);
```

- **Parameters**

af:
[In] Address family. *AF_INET* for IPv4, *AF_INET6* for IPv6.

src:
[In] IPv4 or IPv6 address string (NULL terminated).

dst:
[Out] Resulting IPv4/IPv6 address.

- **Return Value**

0 This function is executed successfully.
1 The address format is bad.
-1 af is not *AF_INET* or *AF_INET6*.

- **Dependencies**

None.

5.2.3. inet_ntop

Formats an IPv4/IPv6 address into a NULL-terminated string.

- **Prototype**

```
const char* inet_ntop ( int32_t af, const void * src, char * dst, size_t size );
```

- **Parameters**

af:
[In] Address family. *AF_INET* for IPv4, *AF_INET6* for IPv6.

src:
[In] IPv4 or IPv6 address string (NULL terminated).

dst:
[Out] Resulting IPv4/IPv6 address.

size:
[Out] Size of the output buffer in bytes.

- **Return Value**

Pointer to the resulting string if OK, else NULL.

- **Dependencies**

None.

5.2.4. qapi_Net_Interface_Get_Physical_Address

Retrieves the physical address and physical address length of an interface.

Note that all arguments must not be 0. Also note that this function does not allocate space for the address, and therefore the caller must not free it.

- **Prototype**

```
int32_t qapi_Net_Interface_Get_Physical_Address ( const char * interface_Name, const uint8_t ** address, uint32_t * address_Len );
```

- **Parameters**

interface_Name:
[In] Name of the interface for which to retrieve the physical address and or physical address length.

address:
[Out] Pointer to where to save the address of the buffer containing the physical address.

address_Len:
[Out] Pointer to where to store the physical address length.

- **Return Value**

0 is returned on success.
A negative error code is returned on failure.

- **Dependencies**

None.

5.2.5. qapi_Net_Interface_Exist

Checks whether the interface exists.

- **Prototype**

```
qbool_t qapi_Net_Interface_Exist ( const char * interface_Name );
```

- **Parameters**

interface_Name:
[in] Name of the interface for which to check whether it exists.

- **Return Value**

0 The interface does not exist.
1 The interface does exist.

- **Dependencies**

None.

5.2.6. qapi_Net_IPv4_Config

Sets/gets IPv4 parameters, or triggers the DHCP client.

- **Prototype**

```
qapi_Status_t qapi_Net_IPv4_Config (const char * interface_Name, qapi_Net_IPv4cfg_Command_t cmd, uint32_t * ipv4_Addr, uint32_t * subnet_Mask, uint32_t * gateway );
```

- **Parameters**

interface_Name:
[In] Name of the interface for which to check whether it exists.

cmd:
[In] Command mode. Possible values are:

QAPI_NET_IPv4CFG_QUERY_E	(0)	Get the IPv4 parameters of an interface.
QAPI_NET_IPv4CFG_STATIC_IP_E	(1)	Assign the IPv4 address, subnet mask, and default gateway.

ipv4_Addr:
[In] Pointer to the IPv4 address in host order.

subnet_Mask:
[In] Pointer to the IPv4 subnet mask in host order.

gateway:
[In] Pointer to the IPv4 gateway address in host order.

- **Return Value**

0 This function is executed successfully.
1 It fails to execute this function.

- **Dependencies**

None.

5.2.7. qapi_Net_Ping

Sends an IPv4 ping.

- **Prototype**

```
qapi_Status_t qapi_Net_Ping ( uint32_t ipv4_Addr, uint32_t size );
```

- **Parameters**

ipv4_Addr:
[In] IPv4 destination address in network order.

size:
[In] Size of the ping payload in bytes.

- **Return Value**

0 This function is executed successfully.
1 It fails to execute this function.

- **Dependencies**

None.

5.2.8. qapi_Net_Ping_2

Sends an IPv4 ping request.

- **Prototype**

```
qapi_Status_t qapi_Net_Ping_2 ( qapi_Net_Ping_V4_t * ping_buf, qapi_Ping_Info_Resp_t * ping_resp );
```

- **Parameters**

ping_buf:
[In] Pointer to IPv4 ping structure. The structure will take the IPv4 destination address in network order, the IPv4 address to which to send the ping via this source, the number of data bytes to send, and a Ping request timeout value (in ms).

ping_resp:
[Out] Pointer to where to store the ping response code and the type for the ICMP echo response received.

- **Return Value**

QAPI_OK Successful ping response is received.
QAPI_ERROR The response buffer is filled with an error code.

- **Dependencies**

None.

5.2.9. qapi_Net_Ping_3

Sends an IPv4 ping request. Compared to *qapi_Net_Ping_2()*, parameter *ttl* was added.

- **Prototype**

```
qapi_Status_t qapi_Net_Ping_3 ( qapi_Net_Ping_V4_R2_t *ping_buf, qapi_Ping_Info_Resp_R2_t * ping_resp );
```

- **Parameters**

ping_buf:
[In] Pointer to IPv4 ping structure. The structure will take the IPv4 destination address in network order, the IPv4 address to which to send the ping via this source, the number of data bytes to send, and a Ping request timeout value (in ms).

ping_resp:
[Out] Pointer to where to store the ping response code and the type for the ICMP echo response received.

- **Return Value**

QAPI_OK	Successful ping response is received.
QAPI_ERROR	The response buffer is filled with an error code.

- **Dependencies**

None.

5.2.10. qapi_Net_IPv4_Route

Adds, deletes, or queries the IPv4 route.

- **Prototype**

```
qapi_Status_t qapi_Net_IPv4_Route ( const char * interface_Name, qapi_Net_Route_Command_t cmd, uint32_t * ipv4_Addr, uint32_t * subnet_Mask, uint32_t * gateway, qapi_Net_IPv4_Route_List_t * route_List );
```

- **Parameters**

interface_Name:
[In] Pointer to the interface name.

cmd:
[Out] Command mode. Possible values are:

QAPI_NET_ROUTE_ADD_E	(0)	Add route
QAPI_NET_ROUTE_DEL_E	(1)	Delete route
QAPI_NET_ROUTE_SHOW_E	(2)	Show route

ipv4_Addr:
[In] Pointer to the IPv4 address in host order.

subnet_Mask:
[In] Pointer to the IPv4 subnet mask in host order.

gateway:
[In] Pointer to the IPv4 gateway address in host order.

route_List:
[In] Pointer to the buffer to contain the list of routes, returned with the *QAPI_NET_ROUTE_SHOW_E* command.

- **Return Value**

0 This function is executed successfully.
1 It fails to execute this function.

- **Dependencies**

None.

5.2.11. qapi_Net_Ping6

Sends an IPv6 ping request.

- **Prototype**

```
qapi_Status_t qapi_Net_Ping6 ( uint8_t ipv6_Addr[16], uint32_t size, const char * interface_Name );
```

- **Parameters**

ipv6_Addr:
[In] IPv6 address to which to send a ping.

size:
[In] Number of data bytes to send.

interface_Name:
[In] Pointer to the interface name; the interface name is required when pinging an IPv6 link local address.

- **Return Value**

0 Ping response is received.
1 Ping request timed out.
-1 Error.

- **Dependencies**

None.

5.2.12. qapi_Net_Ping6_2

Sends an IPv6 ping request with a response.

- **Prototype**

```
qapi_Status_t qapi_Net_Ping6_2 ( qapi_Net_Ping_V6_t * ping6_buf, qapi_Ping_Info_Resp_t * ping_resp );
```

- **Parameters**

ping6_buf:
[In] Pointer to the IPv6 ping structure. The structure will take the IPv6 address to which to send a ping, the IPv6 address to send the ping via this source, the number of data bytes to send, the ping request timeout value (in ms), and when pinging an IPv6 link local address interface, a name is required.

ping_resp:
[In] Pointer to where to store the ping response code and the type for the ICMP echo response received.

- **Return Value**

QAPI_OK	A successful ping response is received.
QAPI_ERROR	The error and response buffer is filled with the error code.

- **Dependencies**

None.

5.2.13. qapi_Net_Ping6_3

Sends an IPv6 ping request with a response. Compared to *qapi_Net_Ping6_2()*, parameter *ttl* was added.

- **Prototype**

```
qapi_Status_t qapi_Net_Ping6_3(qapi_Net_Ping_V6_R2_t *ping6_buf, qapi_Ping_Info_Resp_R2_t *ping_resp);
```

- **Parameters**

ping6_buf:
[In] Pointer to the IPv6 ping structure. The structure will take the IPv6 address to which to send a ping, the IPv6 address to send the ping via this source, the number of data bytes to send, the ping request timeout value (in ms), and when pinging an IPv6 link local address interface, a name is required.

ping_resp:
[In] Pointer to where to store the ping response code and the type for the ICMP echo response received.

- **Return Value**

QAPI_OK A successful ping response is received.
QAPI_ERROR The error and response buffer is filled with the error code.

- **Dependencies**

None.

5.2.14. qapi_Net_IPv6_Get_Address

Gets the IPv6 addresses of an interface.

- **Prototype**

```
qapi_Status_t qapi_Net_IPv6_Get_Address ( const char * interface_Name, uint8_t * link_Local, uint8_t * global, uint8_t * default_Gateway, uint8_t * global_Second, uint32_t * link_Local_Prefix, uint32_t * global_Prefix, uint32_t * default_Gateway_Prefix, uint32_t * global_Second_Prefix );
```

- **Parameters**

interface_Name:
[In] Pointer to the name of the network interface.

link_Local:
[In] Pointer to the first global unicast address.

global:
[In] Pointer to the link local unicast address.

default_Gateway:
[In] Pointer to the default gateway address.

global_Second:
[In] Pointer to the second global unicast address.

link_Local_Prefix:
[In] Pointer to the prefix length of the link-local address.

global_Prefix:
[In] Pointer to the prefix length of the first global address.

default_Gateway_Prefix:
[In] Pointer to the prefix length of the default gateway address.

global_Second_Prefix:

[In] Pointer to the prefix length of the second global address.

- **Return Value**

0 This function is executed successfully.
1 It fails to execute this function.

- **Dependencies**

None.

5.2.15. qapi_Net_IPv6_Route

Adds, deletes, or queries the IPv6 route.

- **Prototype**

```
qapi_Status_t qapi_Net_IPv6_Route ( const char * interface_Name, qapi_Net_Route_Command_t cmd, uint8_t * ipv6_Addr, uint32_t * prefix_Length, uint8_t * next_Hop, qapi_Net_IPv6_Route_List_t * route_List );
```

- **Parameters**

interface_Name:
[In] Pointer to the name of the network interface.

cmd:
[In] Command mode. Possible values are:

QAPI_NET_ROUTE_ADD_E	(0)	Add route
QAPI_NET_ROUTE_DEL_E	(1)	Delete route
QAPI_NET_ROUTE_SHOW_E	(2)	Show route

ipv6_Addr:
[In] Pointer to the IPv6 address.

prefix_Length:
[In] Pointer to the IPv6 prefix length.

next_Hop:
[In] Pointer to the IPv6 gateway address.

route_List:
[In] Pointer to the buffer containing a list of routes, returned with the *QAPI_NET_ROUTE_SHOW_E* command.

- **Return Value**

0 This function is executed successfully.
1 It fails to execute this function.

- **Dependencies**

None.

5.2.16. qapi_Net_IPv6_Get_Scope_ID

Returns the scope ID for the interface.

When using link-local addressing with the IPv6 protocol, the scope ID must be specified along with the destination address. The application should use this function to retrieve a scope ID based on the interface name.

- **Prototype**

```
qapi_Status_t qapi_Net_IPv6_Get_Scope_ID ( const char * interface_Name, int32_t * scope_ID );
```

- **Parameters**

interface_Name:
[In] Pointer to the name of the interface for which to retrieve the scope ID.

scope_ID:
[Out] Pointer to the location store the scope ID.

- **Return Value**

0 This function is executed successfully.
1 It fails to execute this function.

- **Dependencies**

None.

6 Domain Name System Client Service APIs

The Domain Name System (DNS) Client service provides a collection of API functions that allow the application to both configure DNS services in the system as well as translate domain names to their numerical IPv4 or IPv6 (or both) addresses, which is needed for the purpose of initiating communications with a remote server or service. The DNS client service can be either manually configured or automatically configured when the DHCP client is enabled.

This chapter provides the following APIs:

```
qapi_Net_DNSc_Is_Started
qapi_Net_DNSc_Command
qapi_Net_DNSc_Reshost
qapi_Net_DNSc_Reshost_on_iface
qapi_Net_DNSc_Get_Server_List
qapi_Net_DNSc_Get_Server_Index
qapi_Net_DNSc_Add_Server
qapi_Net_DNSc_Add_Server_on_iface
qapi_Net_DNSc_Del_Server
qapi_Net_DNSc_Del_Server_on_iface
```

6.1. Data Types

6.1.1. Enumeration Type

6.1.1.1. enum qapi_Net_DNS_Command_t

Commands to start/stop/disable a DNS client.

```
typedef enum
{
    QAPI_NET_DNS_DISABLE_E,
    QAPI_NET_DNS_START_E,
    QAPI_NET_DNS_STOP_E
```

```
} qapi_Net_DNS_Command_t;
```

- **Parameters**

Parameter	Description
QAPI_NET_DNS_DISABLE_E	Stop plus free the space for internal data structures.
QAPI_NET_DNS_START_E	Allocate space for internal data structures; DNS query is allowed after the start command. DNS responses from the server.
QAPI_NET_DNS_STOP_E	Stop sending DNS requests and processing DNS responses; keep internal data structures.

6.1.2. Definition Type

6.1.2.1. The Maximum number of DNS Server

For use with *qapi_Net_DNSc_Get_Server_List()* to get IP addresses of DNS servers.

```
#define MAX_DNS_SVR_NUM 4
```

6.1.2.2. DNS Server Port

```
#define QAPI_DNS_PORT 53
```

6.1.2.3. DNS Server ID

```
#define QAPI_NET_DNS_ANY_SERVER_ID                0xFFFF
#define QAPI_NET_DNS_V4_PRIMARY_SERVER_ID         0
#define QAPI_NET_DNS_V4_SECONDARY_SERVER_ID       1
#define QAPI_NET_DNS_V6_PRIMARY_SERVER_ID         2
#define QAPI_NET_DNS_V6_SECONDARY_SERVER_ID       3
```

- **Parameters**

Parameter	Description
QAPI_NET_DNS_ANY_SERVER_ID	Number of DNS servers in the system, which is a tunable configuration. Use *ANY_SERVER_ID* to populate a free entry, or use an index to update a specific entry.

QAPI_NET_DNS_V4_PRIMARY_SERVER_ID	DNS IPv4 primary server ID.
QAPI_NET_DNS_V4_SECONDARY_SERVER_ID	DNS IPv4 secondary server ID.
QAPI_NET_DNS_V6_PRIMARY_SERVER_ID	DNS IPv6 primary server ID.
QAPI_NET_DNS_V6_SECONDARY_SERVER_ID	DNS IPv6 secondary server ID.

6.1.3. Structure Type

6.1.3.1. struct qapi_Net_DNS_Server_List_t

Use with *qapi_Net_DNSc_Get_Server_List()* to get IP addresses of DNS servers.

```
typedef struct
{
    struct ip46addr svr[MAX_DNS_SVR_NUM];
} qapi_Net_DNS_Server_List_t;
```

- **Parameters**

Type	Parameter	Description
struct ip46addr	*svr*	DNS servers IP addresses.

6.2. API Functions

6.2.1. qapi_Net_DNSc_Is_Started

Check whether the DNS client has started.

- **Prototype**

```
int32_t qapi_Net_DNSc_Is_Started ( void );
```

- **Parameters**

None.

- **Return Value**

0 if not started or 1 if started.

- **Dependencies**

None.

6.2.2. qapi_Net_DNSc_Command

Starts, stops, or disables the DNS client.

- **Prototype**

```
int32_t qapi_Net_DNSc_Command ( qapi_Net_DNS_Command_t cmd );
```

- **Parameters**

cmd:
[In] Command to start/stop/disable the DNS client. The macro of supported commands are *QAPI_NET_DNS_DISABLE_E*, *QAPI_NET_DNS_START_E*, and *QAPI_NET_DNS_STOP_E*.

- **Return Value**

0 This function is executed successfully.
-1 It fails to execute this function.

- **Dependencies**

None.

6.2.3. qapi_Net_DNSc_Reshost

Resolves an IP address text string into an actual IP address.

- **Prototype**

```
int32_t qapi_Net_DNSc_Reshost ( char * hostname, struct ip46addr * ipaddr );
```

- **Parameters**

hostname:
[In] Pointer to an IP address string or host name string.

Ipaddr:
[In, Out] Pointer to struct ip46addr for the resolved IP address. The caller must specify which IP address

(v4 or v6) it intends to resolve to:
If ipaddr type is *AF_INET*, resolve to an IPv4 address.
If ipaddr type is *AF_INET6*, resolve to an IPv6 address.

- **Return Value**

On success, 0 is returned.
On error, a negative number is returned.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

6.2.4. qapi_Net_DNSc_Reshost_on_iface

Resolves an IP address text string into an actual IP address for an interface.

- **Prototype**

```
int32_t qapi_Net_DNSc_Reshost_on_iface ( char * hostname, struct ip46addr* addr, char * iface_index );
```

- **Parameters**

hostname:
[In] Pointer to an IP address string or host name string.

addr:
[In, Out] Pointer to struct ip46addr for the resolved IP address. The caller must specify which IP address (v4 or v6) it intends to resolve to:
If ipaddr type is *AF_INET*, resolve to an IPv4 address.
If ipaddr type is *AF_INET6*, resolve to an IPv6 address.

iface_index:
[In] Name of the PDN/APN for which the address text string is to be resolved.

- **Return Value**

On success, 0 is returned.
On error, A negative number is returned.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

6.2.5. qapi_Net_DNSc_Get_Server_List

Gets the list of configured DNS servers.

- **Prototype**

```
int32_t qapi_Net_DNSc_Get_Server_List(qapi_Net_DNS_Server_List_t*svr_list, uint8_t iface_index );
```

- **Parameters**

svr_list:
[Out] Pointer to a buffer to contain the list.

iface_index:
[In] Index of the configured DNS servers.

- **Return Value**

On success, 0 is returned.
On error, -1 is returned.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

6.2.6. qapi_Net_DNSc_Get_Server_Index

Gets the index at which a DNS server is added to the system.

- **Prototype**

```
qapi_Status_t qapi_Net_DNSc_Get_Server_Index ( char * svr_addr, uint32_t * id, char * iface );
```

- **Parameters**

svr_addr:
[In] Pointer to the DNS server's IP address string.

id:
[Out] Pointer to the server index. This is filled with the position at which *svr_addr* is added.

iface:
[In] Pointer to the interface string on which the server is added.

- **Return Value**

QAPI_OK	This function is executed successfully..
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

6.2.7. qapi_Net_DNSc_Add_Server

Adds a DNS server to the system.

- **Prototype**

```
int32_t qapi_Net_DNSc_Add_Server ( char * svr_addr, uint32_t id );
```

- **Parameters**

svr_addr:
[In] Pointer to the DNS server's IP address string.

id:
[In] Server ID can be *QAPI_NET_DNS_V4_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V4_SECONDARY_SERVER_ID*, *QAPI_NET_DNS_V6_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V6_SECOND-ARY_SERVER_ID*, or *QAPI_NET_DNS_ANY_SERVER_ID*.

- **Return Value**

On success, 0 is returned.
On error, -1 is returned.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

6.2.8. qapi_Net_DNSc_Add_Server_on_iface

Adds a DNS server to a PDN interface.

- **Prototype**

```
int32_t qapi_Net_DNSc_Add_Server_on_iface ( char * svr_addr, uint32_t id, char * iface );
```

- **Parameters**

svr_addr:
[In] Pointer to the DNS server's IP address string.

id:
[In] Server ID can be *QAPI_NET_DNS_V4_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V4_SECONDARY_SERVER_ID*, *QAPI_NET_DNS_V6_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V6_SECOND-ARY_SERVER_ID*, or *QAPI_NET_DNS_ANY_SERVER_ID*.

iface:
[In] Pointer to the name of the PDN on which the server is to be added.

- **Return Value**

On success, 0 is returned.
On error, -1 is returned.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

6.2.9. qapi_Net_DNSc_Del_Server

Removes a DNS server from the system.

- **Prototype**

```
int32_t qapi_Net_DNSc_Del_Server ( uint32_t id );
```

- **Parameters**

id:
[In] Server ID can be *QAPI_NET_DNS_V4_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V4_SECONDARY_SERVER_ID*, *QAPI_NET_DNS_V6_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V6_SECOND-ARY_SERVER_ID*, or *QAPI_NET_DNS_ANY_SERVER_ID*.

- **Return Value**

On success, 0 is returned.
On error, -1 is returned.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

6.2.10. qapi_Net_DNSc_Del_Server_on_iface

Removes a DNS server from an interface.

- **Prototype**

```
int32_t qapi_Net_DNSc_Del_Server_on_iface ( uint32_t id, char * iface_index );
```

- **Parameters**

id:
[In] Server ID can be *QAPI_NET_DNS_V4_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V4_SECONDARY_SERVER_ID*, *QAPI_NET_DNS_V6_PRIMARY_SERVER_ID*, *QAPI_NET_DNS_V6_SECOND-ARY_SERVER_ID*, or *QAPI_NET_DNS_ANY_SERVER_ID*.

iface_index:
[In] Name of interface from which to delete a DNS server.

- **Return Value**

On success, 0 is returned.
On error, -1 is returned.

- **Dependencies**

DNS client must be started via *qapi_Net_DNSc_Command()*.

7 HTTP(S) APIs

The HTTP client service provides a collection of API functions that allow the application to enable and configure HTTP client services. The HTTP client can be configured to support IPv4, IPv6, as well as HTTP mode, HTTPS mode (secure), or both.

This chapter provides the following APIs:

```
qapi_Net_HTTPc_Start
qapi_Net_HTTPc_Stop
qapi_Net_HTTPc_New_sess
qapi_Net_HTTPc_Free_sess
qapi_Net_HTTPc_Connect
qapi_Net_HTTPc_Proxy_Connect
qapi_Net_HTTPc_Disconnect
qapi_Net_HTTPc_Request
qapi_Net_HTTPc_Set_Body
qapi_Net_HTTPc_Set_Param
qapi_Net_HTTPc_Add_Header_Field
qapi_Net_HTTPc_Clear_Header
qapi_Net_HTTPc_Configure_SSL
qapi_Net_HTTPc_Configure
```

7.1. Data Types

7.1.1. Enumeration Type

7.1.1.1. enum qapi_Net_HTTPc_Method_e

HTTP request types supported by *qapi_Net_HTTPc_Request()*.

```
typedef enum
{
  QAPI_NET_HTTP_CLIENT_GET_E = 1,
  QAPI_NET_HTTP_CLIENT_POST_E,
  QAPI_NET_HTTP_CLIENT_PUT_E,
```

```
    QAPI_NET_HTTP_CLIENT_PATCH_E,
    QAPI_NET_HTTP_CLIENT_HEAD_E,
    QAPI_NET_HTTP_CLIENT_CONNECT_E
} qapi_Net_HTTPc_Method_e;
```

- **Parameters**

Parameter	Description
QAPI_NET_HTTP_CLIENT_GET_E	HTTP get request.
QAPI_NET_HTTP_CLIENT_POST_E	HTTP post request.
QAPI_NET_HTTP_CLIENT_PUT_E	HTTP put request.
QAPI_NET_HTTP_CLIENT_PATCH_E	HTTP patch request.
QAPI_NET_HTTP_CLIENT_HEAD_E	HTTP head request.
QAPI_NET_HTTP_CLIENT_CONNECT_E	HTTP connect request.

7.1.1.2. enum qapi_Net_HTTPc_CB_State_e

HTTP callback state returned by *qapi_HTTPc_CB_t()*.

```
typedef enum
{
    QAPI_NET_HTTPC_RX_ERROR_SERVER_CLOSED = -8,
    QAPI_NET_HTTPC_RX_ERROR_RX_PROCESS = -7,
    QAPI_NET_HTTPC_RX_ERROR_RX_HTTP_HEADER = -6,
    QAPI_NET_HTTPC_RX_ERROR_INVALID_RESPONSECODE = -5,
    QAPI_NET_HTTPC_RX_ERROR_CLIENT_TIMEOUT = -4,
    QAPI_NET_HTTPC_RX_ERROR_NO_BUFFER = -3,
    QAPI_NET_HTTPC_RX_CONNECTION_CLOSED = -2,
    QAPI_NET_HTTPC_RX_ERROR_CONNECTION_CLOSED = -1,
    QAPI_NET_HTTPC_RX_FINISHED = 0,
    QAPI_NET_HTTPC_RX_MORE_DATA = 1,
}qapi_Net_HTTPc_CB_State_e;
```

- **Parameters**

Parameter	Description
QAPI_NET_HTTPC_RX_ERROR_SERVER_CLOSED	HTTP response error – the server

	closed the connection.
QAPI_NET_HTTPC_RX_ERROR_RX_PROCESS	HTTP response error – response is processing.
QAPI_NET_HTTPC_RX_ERROR_RX_HTTP_HEADER	HTTP response error – header is processing.
QAPI_NET_HTTPC_RX_ERROR_INVALID_RESPONSECODE	HTTP response error – invalid response code.
QAPI_NET_HTTPC_RX_ERROR_CLIENT_TIMEOUT	HTTP response error – timeout waiting for a response.
QAPI_NET_HTTPC_RX_ERROR_NO_BUFFER	HTTP response error – memory is unavailable.
QAPI_NET_HTTPC_RX_CONNECTION_CLOSED	HTTP response – connection is closed.
QAPI_NET_HTTPC_RX_ERROR_CONNECTION_CLOSED	HTTP response error – connection is closed.
QAPI_NET_HTTPC_RX_FINISHED	HTTP response – response was received completely.
QAPI_NET_HTTPC_RX_MORE_DATA	HTTP response – there is more response data to be received.

7.1.2. Typedefs

7.1.2.1. typedef void (∗ qapi_HTTPc_CB_t)(void ∗arg, int32_t state, void ∗value)

HTTP response user callback registered during *qapi_Net_HTTPc_New_sess()*.

- **Prototype**

```
typedef void (*qapi_HTTPc_CB_t)
(
   void* arg,
   int32_t state,
   void* value
);
```

- **Paramerters**

arg:
[In] User payload information.

state:
[Out] HTTP response state.

value:
[Out] HTTP response information.

- **Return Value**

None.

7.1.3. Structure Type

7.1.3.1. struct qapi_Net_HTTPc_Response_t

HTTP response data returned by *qapi_HTTPc_CB_t()*.

```
typedef struct
{
   uint32_t         length;
   uint32_t         resp_Code;
   const void     *data;
   const void     *rsp_hdr;
   uint32_t         rsp_hdr_len;
} qapi_Net_HTTPc_Response_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*length*	HTTP response data buffer length.
uint32_t	*resp_Code*	HTTP response code.
const void*	*data*	HTTP response data.
const void*	*rsp_hdr*	HTTP response data header.
uint32_t	*rsp_hdr_len*	HTTP response data header length.

7.1.3.2. struct qapi_Net_HTTPc_Sock_Opts_t

Structure to configure an HTTP client session.

```
typedef struct __qapi_Net_HTTPc_Sock_Opts_s
{
   int32_t level;
```

```
    int32_t opt_name;
    void *opt_value;
    uint32_t opt_len;
} qapi_Net_HTTPc_Sock_Opts_t;
```

- **Parameters**

Type	Parameter	Description
int32_t	*level*	Specifies the protocol level at which the option resides.
int32_t	*opt_name*	Socket option name.
void*	*opt_value*	Socket option value.
uint32_t	*opt_len*	Socket option length.

7.1.3.3. struct qapi_Net_HTTPc_Config_t

Structure to configure an HTTP client session.

```
typedef struct __qapi_Net_HTTPc_Config_s
{
    uint16_t addr_type;
    uint32_t sock_options_cnt;
    qapi_Net_HTTPc_Sock_Opts_t *sock_options;
    uint16_t max_send_chunk;
    uint16_t max_send_chunk_delay_ms;
    uint8_t max_send_chunk_retries;
    uint8_t max_conn_poll_cnt;
    uint32_t max_conn_poll_interval_ms;
} qapi_Net_HTTPc_Config_t;
```

- **Parameters**

Type	Parameter	Description
uint16_t	*addr_type*	Address type *AF_UNSPEC*, *AF_INET* or *AF_INET6* (used for DNS resolution only).
uint32_t	*sock_options_cnt*	Number of socket options.
qapi_Net_HTTPc_Sock_Opts_t *	*sock_options*	Socket options -- only the Linger option is currently supported.

uint16_t	*max_send_chunk*	Maximum send data chunk per transaction.
uint16_t	*max_send_chunk_delay_ms*	Maximum delay between send data chunks (msec).
uint8_t	*max_send_chunk_retries*	Maximum send data chunk retries.
uint8_t	*max_conn_poll_cnt*	Maximum connect polling count.
uint32_t	*max_conn_poll_interval_ms*	Maximum connect polling interval

7.2. API Functions

7.2.1. qapi_Net_HTTPc_Start

Starts or restarts an HTTP client module.

This function is invoked to start or restart the HTTP client after it is stopped via a call to *qapi_Net_HTTPc_Stop()*.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Start ( void );
```

- **Parameters**

None.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

None.

7.2.2. qapi_Net_HTTPc_Stop

Stops an HTTP client module.

This function is invoked to stop the HTTP client after it was started via a call to *qapi_Net_HTTPc_Start()*.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Stop ( void );
```

- **Parameters**

None.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*.

7.2.3. qapi_Net_HTTPc_New_sess

Creates a new HTTP client session.

To create a client session, the caller must invoke this function and the handle to the newly created context is returned if successful. As part of the function call, a user callback function is registered with the HTTP client module that gets invoked for that particular session if there is some response data from the HTTP server. Passing in the SSL context information ensures that a secure session is created.

- **Prototype**

```
qapi_Net_HTTPc_handle_t qapi_Net_HTTPc_New_sess ( uint32_t timeout, qapi_Net_SSL_Obj_Hdl_t ssl_Object_Handle, qapi_HTTPc_CB_t callback, void * arg, uint32_t httpc_Max_Body_Length, uint32_t httpc_Max_Header_Length );
```

- **Parameters**

timeout:
[In] Timeout (in ms) of a session method (zero is not recommended).

ssl_Object_Handle:
[In] SSL context for HTTPs connect (zero for no HTTPs session support).

callback:
[In] Register a callback function; NULL for no support for a callback.

arg:
[In] User data payload to be returned by the callback function.

httpc_Max_Body_Length:
[In] Maximum body length for this session.

httpc_Max_Header_Length:
[In] Maximum header length for this session.

- **Return Value**

The handle to the newly created context is returned on success.
NULL otherwise.

- **Dependencies**

HTTP client module must be started via *qapi_Net_HTTPc_Start()*.

7.2.4. qapi_Net_HTTPc_Free_sess

Releases an HTTP client session.

An HTTP client session that is connected to the HTTP server is disconnected before releasing the resources associated with that session.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Free_sess ( qapi_Net_HTTPc_handle_t handle );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client session must be created via *qapi_Net_HTTPc_New_sess()*.

7.2.5. qapi_Net_HTTPc_Connect

Connects an HTTP client session to the HTTP server.

The HTTP client session is connected to the HTTP server in blocking mode.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Connect ( qapi_Net_HTTPc_handle_t handle, const char * URL,
uint16_t port );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

URL:
[In] Server URL informtion.

port:
[In] Server port information.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()* and HTTP client session must be created via *qapi_Net_HTTPc_New_sess()*.

7.2.6. qapi_Net_HTTPc_Proxy_Connect

Connects an HTTP client session to the HTTP proxy server.

The HTTP client session is connected to the HTTP server in blocking mode.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Proxy_Connect ( qapi_Net_HTTPc_handle_t handle, const char *
URL, uint16_t port, uint8_t secure_proxy );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

URL:
[In] Server URL informtion.

port:

[In] Server port information.

secure_proxy:
[In] Secure proxy connection.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()* and HTTP client session must be created via *qapi_Net_HTTPc_New_sess()*.

7.2.7. qapi_Net_HTTPc_Disconnect

Disconnects an HTTP client session from the HTTP server.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Disconnect ( qapi_Net_HTTPc_handle_t handle );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()* and HTTP client session must be created via *qapi_Net_HTTPc_New_sess()*.

7.2.8. qapi_Net_HTTPc_Request

Processes the HTTP client session requests.

HTTP client session requests are processed and sent to the HTTP server.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Request ( qapi_Net_HTTPc_handle_t handle,
qapi_Net_HTTPc_Method_e cmd, const char * URL );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

cmd:
[In] HTTP request method information.

URL:
[In] Server URL information.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*, HTTP client session must be created via *qapi_Net_HTTPc_New_sess()* and connected to HTTP server via *qapi_Net_HTTPc_Connect()*.

7.2.9. qapi_Net_HTTPc_Set_Body

Sets an HTTP client session body.

Multiple invocations of this function will result in overwriting the internal data buffer with the content of the last call.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Set_Body ( qapi_Net_HTTPc_handle_t handle, const char * body,
uint32_t body_Length );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

body:
[In] HTTP body related information buffer.

body_Length:
[In] HTTP body buffer length.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*, HTTP client session must be created via *qapi_Net_HTTPc_New_sess()* and connected to HTTP server via *qapi_Net_HTTPc_Connect()*.

7.2.10. qapi_Net_HTTPc_Set_Param

Sets an HTTP client session parameter.

Multiple invocations of this function will result in appending the parameter key-value pair information to the internal data buffer.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Set_Param ( qapi_Net_HTTPc_handle_t handle, const char * key, const char * value );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

key:
[In] HTTP key related information.

value:
[In] HTTP value associated with the key.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*, and HTTP client session must be created via *qapi_Net_HTTPc_New_sess()*.

7.2.11. qapi_Net_HTTPc_Add_Header_Field

Adds an HTTP client session header field.

Multiple invocations of this function will result in appending the header type-value pair information to the internal header buffer.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Add_Header_Field ( qapi_Net_HTTPc_handle_t handle, const char * type, const char * value );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

type:
[In] HTTP header type related information.

value:
[In] HTTP value associated with the header type.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*, HTTP client session must be created via *qapi_Net_HTTPc_New_sess()* and connected to HTTP server via *qapi_Net_HTTPc_Connect()*.

7.2.12. qapi_Net_HTTPc_Clear_Header

Clears an HTTP client session header.

Invocation of this function clears the entire content associated with the internal header buffer.

- **Prototype**

```
qapi_Status_t qapi_Net_HTTPc_Clear_Header ( qapi_Net_HTTPc_handle_t handle );
```

- **Parameters**

handle:

[In] Handle to the HTTP client session.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*, HTTP client session must be created via *qapi_Net_HTTPc_New_sess()* and connected to HTTP server via *qapi_Net_HTTPc_Connect()*.

7.2.13. qapi_Net_HTTPc_Configure_SSL

Configures an HTTP client session.

Invocation of this function configures the HTTP client SSL session.

- **Prototype**

```
qapi_Status_tqapi_Net_HTTPc_Configure_SSL(qapi_Net_HTTPc_handle_t handle,qapi_Net_SSL_C-
onfig_t * ssl_Cfg );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

ssl_Cfg:
[In] SSL configuration information.

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*, and HTTP client session must be created via *qapi_Net_HTTPc_New_sess()*.

7.2.14. qapi_Net_HTTPc_Configure

Configures the HTTP client session based on the application requirement.

- **Prototype**

```
qapi_Status_tqapi_Net_HTTPc_Configure(qapi_Net_HTTPc_handle_thandle, qapi_Net_HTTPc_Config_t * httpc_Cfg );
```

- **Parameters**

handle:
[In] Handle to the HTTP client session.

httpc_Cfg:
[In] HTTP client configuration information..

- **Return Value**

0 is returned on success.
Other values on error.

- **Dependencies**

HTTP client must be started via *qapi_Net_HTTPc_Start()*, and HTTP client session must be created via *qapi_Net_HTTPc_New_sess()*.

8 MQTT APIs

The MQTT service provides a collection of API functions that allow the application to enable and configure MQTT services.

This chapter provides the following APIs:

```
qapi_Net_MQTT_New
qapi_Net_MQTT_Destroy
qapi_Net_MQTT_Connect
qapi_Net_MQTT_Disconnect
qapi_Net_MQTT_Publish
qapi_Net_MQTT_Publish_Get_Msg_Id
qapi_Net_MQTT_Subscribe
qapi_Net_MQTT_Unsubscribe
qapi_Net_MQTT_Set_Connect_Callback
qapi_Net_MQTT_Set_Subscribe_Callback
qapi_Net_MQTT_Set_Message_Callback
qapi_Net_MQTT_Set_Publish_Callback
```

8.1. Data Types

8.1.1. Enumeration Type

8.1.1.1. enum QAPI_NET_MQTT_SUBSCRIBE_CBK_MSG

Reason codes for a subscription callback.

```
enum QAPI_NET_MQTT_SUBSCRIBE_CBK_MSG
{
   QAPI_NET_MQTT_SUBSCRIBE_DENIED_E,
   QAPI_NET_MQTT_SUBSCRIBE_GRANTED_E,
   QAPI_NET_MQTT_SUBSCRIBE_MSG_E
};
```

- **Parameters**

Parameter	Description
QAPI_NET_MQTT_SUBSCRIBE_DENIED_E	Subscription is denied by the broker.
QAPI_NET_MQTT_SUBSCRIBE_GRANTED_E	Subscription is granted by the broker.
QAPI_NET_MQTT_SUBSCRIBE_MSG_E	Message was received from the broker.

8.1.1.2. enum QAPI_NET_MQTT_CONNECT_CBK_MSG

Connection callback messages.

```
enum QAPI_NET_MQTT_CONNECT_CBK_MSG
{
  QAPI_NET_MQTT_CONNECT_SUCCEEDED_E,
  QAPI_NET_MQTT_TCP_CONNECT_FAILED_E,
  QAPI_NET_MQTT_SSL_CONNECT_FAILED_E,
  QAPI_NET_MQTT_CONNECT_FAILED_E,
};
```

- **Parameters**

Parameter	Description
QAPI_NET_MQTT_CONNECT_SUCCEEDED_E	MQTT connect succeeded.
QAPI_NET_MQTT_TCP_CONNECT_FAILED_E	TCP connect failed.
QAPI_NET_MQTT_SSL_CONNECT_FAILED_E	SSL connect failed.
QAPI_NET_MQTT_CONNECT_FAILED_E	QAPI_MQTT connect failed.

8.1.1.3. enum QAPI_NET_MQTT_CONN_STATE

Connection states.

```
enum QAPI_NET_MQTT_CONN_STATE
{
  QAPI_NET_MQTT_ST_DORMANT_E,
  QAPI_NET_MQTT_ST_TCP_CONNECTING_E,
```

```
    QAPI_NET_MQTT_ST_TCP_CONNECTED_E,
    QAPI_NET_MQTT_ST_SSL_CONNECTING_E,
    QAPI_NET_MQTT_ST_SSL_CONNECTED_E,
    QAPI_NET_MQTT_ST_MQTT_CONNECTING_E,
    QAPI_NET_MQTT_ST_MQTT_CONNECTED_E,
    QAPI_NET_MQTT_ST_MQTT_TERMINATING_E,
    QAPI_NET_MQTT_ST_SSL_TERMINATING_E,
    QAPI_NET_MQTT_ST_TCP_TERMINATING_E,
    QAPI_NET_MQTT_ST_DYING_E,
    QAPI_NET_MQTT_ST_DEAD_E,
};
```

- **Parameters**

Parameter	Description
QAPI_NET_MQTT_ST_DORMANT_E	MQTT connect succeeded.
QAPI_NET_MQTT_ST_TCP_CONNECTING_E	TCP connect failed.
QAPI_NET_MQTT_ST_TCP_CONNECTED_E	SSL connect failed.
QAPI_NET_MQTT_ST_SSL_CONNECTING_E	SSL is connecting.
QAPI_NET_MQTT_ST_SSL_CONNECTING_E	SSL is connected.
QAPI_NET_MQTT_ST_SSL_CONNECTED_E	MQTT is connecting.
QAPI_NET_MQTT_ST_MQTT_CONNECTING_E	MQTT is connected.
QAPI_NET_MQTT_ST_MQTT_TERMINATING_E	MQTT connection is terminating.
QAPI_NET_MQTT_ST_SSL_TERMINATING_E	SSL connection is terminating.
QAPI_NET_MQTT_ST_TCP_TERMINATING_E	TCP connection is terminating.
QAPI_NET_MQTT_ST_DYING_E	MQTT connection is dying.
QAPI_NET_MQTT_ST_DEAD_E	MQTT connection is dead.

8.1.1.4. enum QAPI_NET_MQTT_MSG_TYPES

MQTT message types.

```
enum QAPI_NET_MQTT_MSG_TYPES
```

```
{
   QAPI_NET_MQTT_CONNECT = 1,
   QAPI_NET_MQTT_CONNACK,
   QAPI_NET_MQTT_PUBLISH,
   QAPI_NET_MQTT_PUBACK,
   QAPI_NET_MQTT_PUBREC,
   QAPI_NET_MQTT_PUBREL,
   QAPI_NET_MQTT_PUBCOMP,
   QAPI_NET_MQTT_SUBSCRIBE,
   QAPI_NET_MQTT_SUBACK,
   QAPI_NET_MQTT_UNSUBSCRIBE,
   QAPI_NET_MQTT_UNSUBACK,
   QAPI_NET_MQTT_PINGREQ,
   QAPI_NET_MQTT_PINGRESP,
   QAPI_NET_MQTT_DISCONNECT,
   QAPI_NET_MQTT_MQTT_NO_RESPONSE_MSG_REQD,
   QAPI_NET_MQTT_INVALID_RESP,
};
```

- **Parameters**

Parameter	Description
QAPI_NET_MQTT_CONNECT	Connect.
QAPI_NET_MQTT_CONNACK	Connection acknowledgement.
QAPI_NET_MQTT_PUBLISH	Publish.
QAPI_NET_MQTT_PUBACK	Publish acknowledgement.
QAPI_NET_MQTT_PUBREC	PubRec.
QAPI_NET_MQTT_PUBREL	PubRel.
QAPI_NET_MQTT_PUBCOMP	PubComp.
QAPI_NET_MQTT_SUBSCRIBE	Subscribe.
QAPI_NET_MQTT_SUBACK	Subscribe acknowledgement.
QAPI_NET_MQTT_UNSUBSCRIBE	Unsubscribe.
QAPI_NET_MQTT_UNSUBACK	Unsubscribe acknowledgement.
QAPI_NET_MQTT_PINGREQ	Ping request.

QAPI_NET_MQTT_PINGRESP	Ping response.
QAPI_NET_MQTT_DISCONNECT	Disconnect.
QAPI_NET_MQTT_MQTT_NO_RESPONSE_MSG_REQD	No response message is required.
QAPI_NET_MQTT_INVALID_RESP	Invalid response.

8.1.2. Definition Type

8.1.2.1. MQTT Length Definition

```
#define QAPI_NET_MQTT_MAX_CLIENT_ID_LEN 128
#define QAPI_NET_MQTT_MAX_TOPIC_LEN 128
```

- **Parameters**

Parameter	Description
QAPI_NET_MQTT_MAX_CLIENT_ID_LEN	Maximum client ID length. The MQTT stack uses the same value.
QAPI_NET_MQTT_MAX_TOPIC_LEN	Maximum topic length.

8.1.3. Typedefs

8.1.3.1. typedef void* qapi_Net_MQTT_Hndl_t

MQTT handle.

8.1.3.2. typedef void (*qapi_Net_MQTT_Connect_CB_t)(qapi_Net_MQTT_Hndl_t mqtt, int32_t reason)

Connect callback typedef.

- **Prototype**

```
typedef void (*qapi_Net_MQTT_Connect_CB_t)
(
qapi_Net_MQTT_Hndl_t mqtt,
int32_t reason
```

```
);
```

- **Parameters**

mqtt:
[In] MQTT handle.

reason:
[Out] Connect callback message.

- **Return Value**

None.

8.1.3.3. typedef void (*qapi_Net_MQTT_Subscribe_CB_t)(qapi_Net_MQTT_Hndl_t mqtt, int32_t reason, const uint8_t* topic, int32_t topic_length, int32_t qos, const void* sid)

Subscribe callback typedef.

- **Prototype**

```
typedef void (*qapi_Net_MQTT_Subscribe_CB_t)
(
    qapi_Net_MQTT_Hndl_t mqtt,
    int32_t reason,
    const uint8_t* topic,
    int32_t topic_length,
    int32_t qos,
    const void* sid
);
```

- **Parameters**

mqtt:
[In] MQTT handle.

reason:
[Out] Subscribe callback message.

topic:
[Out] String to the topic subscribed successfully.

topic_length:
[Out] Length of the topic subscribed successfully.

qos:
[Out] QOS level of the topic configured in subscribe message.

sid:
[Out] Reserved.

- **Return Value**

None.

8.1.3.4. typedef void (*qapi_Net_MQTT_Message_CB_t)(qapi_Net_MQTT_Hndl_t mqtt, int32_t reason, const uint8_t* topic, int32_t topic_length, const uint8_t* msg, int32_t msg_length, int32_t qos, const void* sid);

Message callback typedef.

- **Prototype**

```
typedef void (*qapi_Net_MQTT_Message_CB_t)
(
    qapi_Net_MQTT_Hndl_t mqtt,
    int32_t reason,
    const uint8_t* topic,
    int32_t topic_length,
    const uint8_t* msg,
    int32_t msg_length,
    int32_t qos,
    const void* sid
);
```

- **Parameters**

mqtt:
[In] MQTT handle.

reason:
[Out] Message callback message.

topic:
[Out] String to the topic of the received message.

topic_length:
[Out] Length of the topic of the received message.
msg:
[Out] String to the received message.

msg_length:
[Out] Length of the received message.

qos:
[Out] QOS level of the topic configured in subscribe message.

sid:
[Out] Reserved.

- **Return Value**

None.

8.1.3.5. typedef void (*qapi_Net_MQTT_Publish_CB_t)(qapi_Net_MQTT_Hndl_t mqtt, enum QAPI_NET_MQTT_MSG_TYPES msgtype, int qos, uint16_t msg_id);

Publish callback typedef.

- **Prototype**

```
typedef void (*qapi_Net_MQTT_Publish_CB_t)(qapi_Net_MQTT_Hndl_t mqtt, enum QAPI_NET_MQTT_MSG_TYPES msgtype, int qos, uint16_t msg_id);
```

- **Parameters**

mqtt:
[In] MQTT handle.

msgtype:
[Out] Publish message types.

qos:
[Out] QOS level of the topic configured in publish message.

msg_id:
[Out] Publish message id.

- **Return Value**

None.

8.1.4. Structure Type

8.1.4.1. struct qapi_Net_MQTT_config_s

MQTT configuration.

```
struct qapi_Net_MQTT_config_s
{
   struct sockaddr local;
   struct sockaddr remote;
   bool nonblocking_connect;
   uint8_t client_id[QAPI_NET_MQTT_MAX_CLIENT_ID_LEN];
   int32_t client_id_len;
   uint32_t keepalive_duration;
   uint8_t clean_session;
   uint8_t* will_topic;
   int32_t will_topic_len;
   uint8_t* will_message;
   int32_t will_message_len;
   uint8_t will_retained;
   uint8_t will_qos;
   uint8_t* username;
   int32_t username_len;
   uint8_t* password;
   int32_t password_len;
   uint32_t connack_timed_out_sec;
   uint32_t sock_options_cnt;
   qapi_Net_MQTT_Sock_Opts_t *sock_options;
   qapi_Net_SSL_Config_t ssl_cfg;
   qapi_Net_SSL_CAList_t ca_list;
   qapi_Net_SSL_Cert_t cert;
} ;
typedef struct qapi_Net_MQTT_config_s qapi_Net_MQTT_Config_t;
```

- **Parameters**

Type	Parameter	Description
struct sockaddr	*local*	MQTT client IP address and port number.
struct sockaddr	*remote*	MQTT server IP address and port number.

bool	*nonblocking_connect*	Blocking or nonblocking MQTT connection.
uint8_t	*client_id*	MQTT vlient ID.
int32_t	*client_id_len*	MQTT client ID length.
uint32_t	*keepalive_duration*	Conection keepalive duration in seconds.
uint8_t	*clean_session*	Clean session flag; 0 – No clean session, 1 – clean session.
uint8_t *	*will_topic*	Will topic name.
int32_t	*will_topic_len*	Will topic length.
uint8_t *	*will_message*	Will message.
int32_t	*will_message_len*	Will message length.
uint8_t	*will_retained*	Will retain flag.
uint8_t	*will_qos*	Will QOS.
uint8_t *	*username*	Client username.
int32_t	*username_len*	Client user length.
uint8_t*	*password*	Client password.
int32_t	*password_len*	Client password length.
uint32_t	*connack_timed_out_sec*	Timeout value for which the client waits for the *CONNACK* packet from the server.
uint32_t	*sock_options_cnt*	Number of socket options.
qapi_Net_MQTT_Sock_Opts_t *	*sock_options*	Socket options.
qapi_Net_SSL_Config_t	*ssl_cfg*	SSL configuration.
qapi_Net_SSL_CAList_t	*ca_list*	SSL CA cert details.
qapi_Net_SSL_Cert_t	*cert*	SSL cert details.

8.1.4.2. struct qapi_Net_MQTT_Sock_Opts_t

```
typedef struct __qapi_Net_MQTT_Sock_Opts_s
{
```

```
    int32_t level;
    int32_t opt_name;
    void *opt_value;
    uint32_t opt_len;
}  qapi_Net_MQTT_Sock_Opts_t;
```

- **Parameters**

Type	Parameter	Description
int32_t	*level*	Specifies the protocol level at which the option resides.
int32_t	*opt_name*	Socket option name.
void *	*opt_value*	Socket option value.
uint32_t	*opt_len*	Socket option length.

8.2. API Functions

8.2.1. qapi_Net_MQTT_New

Creates a new MQTT context.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_New ( qapi_Net_MQTT_Hndl_t * hndl );
```

- **Parameters**

hndl:
[Out] Newly created MQTT context.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

None.

8.2.2. qapi_Net_MQTT_Destroy

Destroys an MQTT context.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Destroy ( qapi_Net_MQTT_Hndl_t hndl );
```

- **Parameters**

hndl:
[In] Handle for the MQTT context to be destroyed.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*.

8.2.3. qapi_Net_MQTT_Connect

Connects to an MQTT broker.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Connect ( qapi_Net_MQTT_Hndl_t hndl, const qapi_Net_MQTT_Config_t * config );
```

- **Parameters**

hndl:
[In] MQTT handle.

config:
[In] MQTT client configuration.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*.

8.2.4. qapi_Net_MQTT_Disconnect

Disconnects from an MQTT broker.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Disconnect ( qapi_Net_MQTT_Hndl_t hndl );
```

- **Parameters**

hndl:
[In] MQTT handle.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*.

8.2.5. qapi_Net_MQTT_Publish

Publishes a message to a particular topic.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Publish ( qapi_Net_MQTT_Hndl_t hndl, const uint8_t * topic, const uint8_t * msg, int32_t msg_len, int32_t qos, bool retain);
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] MQTT topic.

msg:
[In] MQTT payload.

msg_len:

[In] MQTT payload length.

qos:
[In] QOS to be used for the message.

retain:
[In] Retain flag.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*, and MQTT client should be connected to the server via *qapi_Net_MQTT_Connect()*.

8.2.6. qapi_Net_MQTT_Publish_Get_Msg_Id

Publishes a message to a particular topic.

Prototype

```
qapi_Status_t qapi_Net_MQTT_Publish_Get_Msg_Id ( qapi_Net_MQTT_Hndl_t hndl, const uint8_t * topic, const uint8_t * msg, int32_t msg_len, int32_t qos, bool retain, uint16_t * msg_id );
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] MQTT topic.

msg:
[In] MQTT payload.

msg_len:
[In] MQTT payload length.

qos:
[In] QOS to be used for the message.
retain:
[In] Retain flag.

msg_id:
[Out] Message ID of the MQTT publish message.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*, and MQTT client should be connected to the server via *qapi_Net_MQTT_Connect()*.

8.2.7. qapi_Net_MQTT_Subscribe

Subscribes to a topic.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Subscribe ( qapi_Net_MQTT_Hndl_t hndl, const uint8_t * topic, int32_t qos );
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] Subscription topic.

qos:
[In] QOS to be used.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*, and MQTT client should be connected to the server via *qapi_Net_MQTT_Connect()*.

8.2.8. qapi_Net_MQTT_Unsubscribe

Unsubscribes from a topic.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Unsubscribe ( qapi_Net_MQTT_Hndl_t hndl, const uint8_t * topic );
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] Topic from which to unsubscribe.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*, MQTT client should be connected to the server via *qapi_Net_MQTT_Connect()*, the topic to be unsubscribed should be subscribed via *qapi_Net_MQTT_subscribed()*.

8.2.9. qapi_Net_MQTT_Set_Connect_Callback

Sets a connect callback, which is invoked when connect is successful.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Set_Connect_Callback ( qapi_Net_MQTT_Hndl_t hndl,
qapi_Net_MQTT_Connect_CB_t callback );
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] Callback to be invoked.

- **Return Value**

QAPI_OK on success.

A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*.

8.2.10. qapi_Net_MQTT_Set_Subscribe_Callback

Sets a subscribe callback, which is invoked when a subscription is granted or denied.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Set_Subscribe_Callback ( qapi_Net_MQTT_Hndl_t hndl,
qapi_Net_MQTT_Subscribe_CB_t callback );
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] Callback to be invoked..

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*.

8.2.11. qapi_Net_MQTT_Set_Message_Callback

Sets a message callback, which is invoked when a message is received for a subscribed topic.

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Set_Message_Callback ( qapi_Net_MQTT_Hndl_t hndl,
qapi_Net_MQTT_Message_CB_t callback );
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] Callback to be invoked.

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*.

8.2.12. qapi_Net_MQTT_Set_Publish_Callback

Sets a publish callback, which is invoked when PUBACK (QOS1)/PUBREC, PUBCOMP(QOS2).

- **Prototype**

```
qapi_Status_t qapi_Net_MQTT_Set_Publish_Callback ( qapi_Net_MQTT_Hndl_t hndl,
qapi_Net_MQTT_Publish_CB_t callback );
```

- **Parameters**

hndl:
[In] MQTT handle.

topic:
[In] Callback to be invoked..

- **Return Value**

QAPI_OK on success.
A negative number on failure.

- **Dependencies**

Handle for the MQTT context must be obtained via *qapi_Net_MQTT_New()*.

9 Device Information APIs

Device information module mainly provides a collection of APIs about getting information of device model, firmware version, network-related parameters, etc.

This chapter provides the following APIs:

```
qapi_Device_Info_Init
qapi_Device_Info_Get
qapi_Device_Info_Set_Callback
qapi_Device_Info_Release
qapi_Device_Info_Reset
```

9.1. Data Types

9.1.1. Enumeration Type

9.1.1.1. enum battery_status

Valid values for battery status.

```
typedef enum
{
  QAPI_DEVICE_INFO_BAT_OV = 0x00,
  QAPI_DEVICE_INFO_BAT_UV = 0x01,
  QAPI_DEVICE_INFO_BAT_MISSING = 0x02,
  QAPI_DEVICE_INFO_BAT_GOOD_HEALTH = 0x03
} battery_status;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_BAT_OV	Over battery voltage.
QAPI_DEVICE_INFO_BAT_UV	Low battery voltage.

QAPI_DEVICE_INFO_BAT_MISSING	Battery is missing.
QAPI_DEVICE_INFO_BAT_GOOD_HEALTH	Battery voltage is good.

9.1.1.2. enum srv_status

Valid values for network service status.

```
typedef enum
{
  QAPI_DEVICE_INFO_SRV_STATE_NO_SRV = 0,
  QAPI_DEVICE_INFO_SRV_STATE_SRV = 1
} srv_status;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_SRV_STATE_NO_SRV	No service.
QAPI_DEVICE_INFO_SRV_STATE_SRV	Service is available.

9.1.1.3. enum nw_indication

Valid values for network indication.

```
typedef enum
{
  QAPI_DEVICE_INFO_NW_IND_NO_SRV = 0,
  QAPI_DEVICE_INFO_NW_IND_SRV = 1
} nw_indication;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_NW_IND_NO_SRV	No service.
QAPI_DEVICE_INFO_NW_IND_SRV	Service is available.

9.1.1.4. enum rrc_state

Valid values for network RRC state.

```
typedef enum
{
   QAPI_DEVICE_INFO_RRC_IDLE = 0,
   QAPI_DEVICE_INFO_RRC_CONNECTED = 1
} rrc_state;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_RRC_IDLE	Status: Idle.
QAPI_DEVICE_INFO_RRC_CONNECTED	Status: connected.

9.1.1.5. enum emm_state

Valid values for network Extended Mobility Management (EMM) state.

```
typedef enum
{
   QAPI_DEVICE_INFO_EMM_NULL = 0,
   QAPI_DEVICE_INFO_EMM_DEREGISTERED = 1,
   QAPI_DEVICE_INFO_EMM_REGISTERED_INITIATED = 2,
   QAPI_DEVICE_INFO_EMM_REGISTERED = 3,
   QAPI_DEVICE_INFO_EMM_TRACKING_AREA_UPDATING_INITIATED = 4,
   QAPI_DEVICE_INFO_EMM_SERVICE_REQUEST_INITIATED = 5,
   QAPI_DEVICE_INFO_EMM_DEREGISTERED_INITIATED = 6
} emm_state;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_EMM_NULL	Null.
QAPI_DEVICE_INFO_EMM_DEREGISTERED	Deregistered.
QAPI_DEVICE_INFO_EMM_REGISTERED_INITIATED	Registered, initiated.

QAPI_DEVICE_INFO_EMM_REGISTERED	Registered.
QAPI_DEVICE_INFO_EMM_TRACKING_AREA_UPDATING_INITIATED	Tracking area update initiated.
QAPI_DEVICE_INFO_EMM_SERVICE_REQUEST_INITIATED	Service request initiated.
QAPI_DEVICE_INFO_EMM_DEREGISTERED_INITIATED	Deregistered, initiated.

9.1.1.6. enum roaming_info

Valid values for network roaming status.

```
typedef enum
{
  QAPI_DEVICE_INFO_ROAMING_STATUS_OFF = 0,
  QAPI_DEVICE_INFO_ROAMING_STATUS_ON = 1
} roaming_info;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_ROAMING_STATUS_OFF	Roaming status: OFF.
QAPI_DEVICE_INFO_ROAMING_STATUS_ON	Roaming status: ON.

9.1.1.7. enum sim_state

Valid value for SI state.

```
typedef enum
{
  QAPI_DEVICE_INFO_SIM_STATE_UNKNOWN = 0x00,
  QAPI_DEVICE_INFO_SIM_STATE_DETECTED = 0x01,
  QAPI_DEVICE_INFO_SIM_STATE_PIN1_OR_UPIN_REQ = 0x02,
  QAPI_DEVICE_INFO_SIM_STATE_PUK1_OR_PUK_REQ = 0x03,
  QAPI_DEVICE_INFO_SIM_STATE_PERSON_CHECK_REQ = 0x04,
  QAPI_DEVICE_INFO_SIM_STATE_PIN1_PERM_BLOCKED = 0x05,
  QAPI_DEVICE_INFO_SIM_STATE_ILLEGAL = 0x06,
  QAPI_DEVICE_INFO_SIM_STATE_READY = 0x07
} sim_state;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_SIM_STATE_UNKNOWN	Unknown.
QAPI_DEVICE_INFO_SIM_STATE_DETECTED	Detected.
QAPI_DEVICE_INFO_SIM_STATE_PIN1_OR_UPIN_REQ	PIN1 or UPIN is required.
QAPI_DEVICE_INFO_SIM_STATE_PUK1_OR_PUK_REQ	PUK1 or PUK for UPIN is required.
QAPI_DEVICE_INFO_SIM_STATE_PERSON_CHECK_REQ	Personalization state must be checked.
QAPI_DEVICE_INFO_SIM_STATE_PIN1_PERM_BLOCKED	PIN1 is blocked.
QAPI_DEVICE_INFO_SIM_STATE_ILLEGAL	Illegal.
QAPI_DEVICE_INFO_SIM_STATE_READY	Ready.

9.1.1.8. enum qapi_Device_Info_Type_t

Device information types.

```
typedef enum
{
  QAPI_DEVICE_INFO_BUILD_ID_E,
  QAPI_DEVICE_INFO_IMEI_E,
  QAPI_DEVICE_INFO_IMSI_E,
  QAPI_DEVICE_INFO_OS_VERSION_E,
  QAPI_DEVICE_INFO_MANUFACTURER_E,
  QAPI_DEVICE_INFO_MODEL_ID_E,
  QAPI_DEVICE_INFO_BATTERY_STATUS_E,
  QAPI_DEVICE_INFO_BATTERY_PERCENTAGE_E,
  QAPI_DEIVCE_INFO_TIME_ZONE_E,
  QAPI_DEIVCE_INFO_ICCID_E,
  QAPI_DEVICE_INFO_4G_SIG_STRENGTH_E,
  QAPI_DEVICE_INFO_BASE_STATION_ID_E,
  QAPI_DEVICE_INFO_MCC_E,
  QAPI_DEVICE_INFO_MNC_E,
  QAPI_DEVICE_INFO_SERVICE_STATE_E,
  QAPI_DEVICE_INFO_MDN_E,
  QAPI_DEVICE_INFO_TAC_E,
  QAPI_DEVICE_INFO_CELL_ID_E,
  QAPI_DEVICE_INFO_RCCS_E,
```

```
  QAPI_DEVICE_INFO_EMMS_E,
  DEPRACATED1,
  QAPI_DEVICE_INFO_SERVING_PCI_E,
  QAPI_DEVICE_INFO_SERVING_RSRQ_E,
  QAPI_DEVICE_INFO_SERVING_EARFCN_E,
  DEPRACATED2,
  DEPRACATED3,
  DEPRACATED4,
  DEPRACATED5,
  DEPRACATED6,
  QAPI_DEVICE_INFO_NETWORK_IND_E,
  QAPI_DEVICE_INFO_ROAMING_E,
  QAPI_DEVICE_INFO_LAST_POWER_ON_E,
  QAPI_DEVICE_INFO_CHIPID_STRING_E,
  QAPI_DEVICE_INFO_APN_PROFILE_INDEX_E,
  QAPI_DEVICE_INFO_SIM_STATE_E,
  QAPI_DEVICE_INFO_NETWORK_BEARER_E,
  QAPI_DEVICE_INFO_LINK_QUALITY_E,
  QAPI_DEVICE_INFO_TX_BYTES_E,
  QAPI_DEVICE_INFO_RX_BYTES_E,
  QAPI_DEVICE_INFO_ANY,
} qapi_Device_Info_ID_t;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_BUILD_ID_E	Device BUILD_ID.
QAPI_DEVICE_INFO_IMEI_E	Device IMEI.
QAPI_DEVICE_INFO_IMSI_E	UIM IMSI.
QAPI_DEVICE_INFO_OS_VERSION_E	Device OS version.
QAPI_DEVICE_INFO_MANUFACTURER_E	Device manufacturer.
QAPI_DEVICE_INFO_MODEL_ID_E	Device model ID.
QAPI_DEVICE_INFO_BATTERY_STATUS_E	Device battery status.
QAPI_DEVICE_INFO_BATTERY_PERCENTAGE_E	Device battery percentage.
QAPI_DEIVCE_INFO_TIME_ZONE_E	Device time zone.
QAPI_DEIVCE_INFO_ICCID_E	Device ICCID.

QAPI_DEVICE_INFO_4G_SIG_STRENGTH_E	Network signal strength.
QAPI_DEVICE_INFO_BASE_STATION_ID_E	Network base station ID.
QAPI_DEVICE_INFO_MCC_E	Network MCC.
QAPI_DEVICE_INFO_MNC_E	Network MNC.
QAPI_DEVICE_INFO_SERVICE_STATE_E	Network service status.
QAPI_DEVICE_INFO_MDN_E	Device MDN.
QAPI_DEVICE_INFO_TAC_E	Network tracking area code.
QAPI_DEVICE_INFO_CELL_ID_E	Network cell ID.
QAPI_DEVICE_INFO_RCCS_E	Network RRC state.
QAPI_DEVICE_INFO_EMMS_E	Network EMM state.
DEPRACATED1	Information to keep enum numbering consistent.
QAPI_DEVICE_INFO_SERVING_PCI_E	Network serving cell PCI.
QAPI_DEVICE_INFO_SERVING_RSRQ_E	Serving cell RSRQ.
QAPI_DEVICE_INFO_SERVING_EARFCN_E	Serving cell EARFCN.
DEPRACATED2	Information to keep enum numbering consistent.
DEPRACATED3	Information to keep enum numbering consistent.
DEPRACATED4	Information to keep enum numbering consistent.
DEPRACATED5	Information to keep enum numbering consistent.
DEPRACATED6	Information to keep enum numbering consistent.
QAPI_DEVICE_INFO_NETWORK_IND_E	Network indication.
QAPI_DEVICE_INFO_ROAMING_E	Roaming status.
QAPI_DEVICE_INFO_LAST_POWER_ON_E	Last power on time.
QAPI_DEVICE_INFO_CHIPID_STRING_E	Chipset name.
QAPI_DEVICE_INFO_APN_PROFILE_INDEX_E	APN profile index.

QAPI_DEVICE_INFO_SIM_STATE_E	SIM state.
QAPI_DEVICE_INFO_NETWORK_BEARER_E	Network bearer.
QAPI_DEVICE_INFO_LINK_QUALITY_E	Network link quality.
QAPI_DEVICE_INFO_TX_BYTES_E	Device Tx bytes.
QAPI_DEVICE_INFO_RX_BYTES_E	Device Rx bytes.
QAPI_DEVICE_INFO_ANY	Any device information.

9.1.1.9. enum qapi_Device_Info_Type_t

Device information response types.

```
typedef enum
{
  QAPI_DEVICE_INFO_TYPE_BOOLEAN_E,
  QAPI_DEVICE_INFO_TYPE_INTEGER_E,
  QAPI_DEVICE_INFO_TYPE_BUFFER_E,
} qapi_Device_Info_Type_t;
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_TYPE_BOOLEAN_E	Response type is Boolean.
QAPI_DEVICE_INFO_TYPE_INTEGER_E	Response type is integer.
QAPI_DEVICE_INFO_TYPE_BUFFER_E	Response type is buffer.

9.1.2. Definition Type

9.1.2.1. Macro for Network Bearer Values

```
#define QAPI_DEVICE_INFO_SRV_TYPE_LTE      1
#define QAPI_DEVICE_INFO_SRV_TYPE_GSM      2
#define QAPI_DEVICE_INFO_SRV_TYPE_WCDMA    3
#define QAPI_DEVICE_INFO_LTE_TDD           5
#define QAPI_DEVICE_INFO_LTE_FDD           6
```

```
#define QAPI_DEVICE_INFO_LTE_NB_IOT 7
```

- **Parameters**

Parameter	Description
QAPI_DEVICE_INFO_SRV_TYPE_LTE	Nw bearer srvc type: LTE.
QAPI_DEVICE_INFO_SRV_TYPE_GSM	Nw bearer srvc type: GSM.
QAPI_DEVICE_INFO_SRV_TYPE_WCDMA	Nw bearer srvc type: WCDMA.
QAPI_DEVICE_INFO_LTE_TDD	Nw bearer srvc type: LTE Mode: TDD.
QAPI_DEVICE_INFO_LTE_FDD	Nw bearer srvc type: LTE Mode: FDD.
QAPI_DEVICE_INFO_LTE_NB_IOT	Nw bearer srvc type: LTE Mode: NB-IOT.

9.1.2.2. Maximum Size of valuebuf of Structure qapi_Device_Info_t.

```
#define QAPI_DEVICE_INFO_BUF_SIZE 128
```

9.1.3. Typedefs

9.1.3.1. typedef void (*qapi_Device_Info_Callback_t_v2)(qapi_Device_Info_Hndl_t device_info_hndl, const qapi_Device_Info_t *info)

QAPI Device Info Callback.

- **Prototype**

```
typedef void (*qapi_Device_Info_Callback_t_v2)(qapi_Device_Info_Hndl_t device_info_hndl, const qapi_Device_Info_t *info);
```

- **Parameters**

device_info_hndl:
[In] Handle for the device information context.

info:
[Out] information reveived for the specified information ID via *qapi_Device_Info_Set_Callback()*.

- **Return Value**

None.

9.1.4. Structure Type

9.1.4.1. struct qapi_Device_Info_t

QAPI device information structure.

```
typedef struct
{
   qapi_Device_Info_ID_t id;
   qapi_Device_Info_Type_t info_type;
   union
   {
      struct
      {
         char buf[QAPI_DEVICE_INFO_BUF_SIZE];
         uint32_t len;
      } valuebuf;
      int64_t valueint;
      bool valuebool;
   } u;
} qapi_Device_Info_t;
```

- **Parameters**

Type	Parameter	Description
qapi_Device_Info_ID_t	*id*	Required information ID.
qapi_Device_Info_Type_t	*info_type*	Response type.
union qapi_Device_Info_t	*u*	Union of values.

9.1.4.2. union qapi_Device_Info_t.u

```
union
{
struct
{
```

```
        char buf[QAPI_DEVICE_INFO_BUF_SIZE];
        uint32_t len;
    } valuebuf;
    int64_t valueint;
    bool valuebool;
}u;
```

- **Parameters**

Type	Parameter	Description
u	*valuebuf*	Union of values. Union of buffer values.
int	*valueint*	Response integer value.
bool	*valuebool*	Response Boolean value.

9.1.4.3. struct qapi_Device_Info_t.u.valuebuf

```
struct
{
   char buf[QAPI_DEVICE_INFO_BUF_SIZE];
   uint32_t len;
} valuebuf;
```

- **Parameters**

Type	Parameter	Description
char	*buf*	Response buffer.
uint32_t	*len*	Length of the response string.

9.2. API Functions

9.2.1. qapi_Device_Info_Init_v2

Initializes the device information context.

This function must be called before invoking other qapi_Device_Info APIs.

- **Prototype**

```
qapi_Status_t qapi_Device_Info_Init_v2(qapi_Device_Info_Hndl_t *device_info_hndl);
```

- **Parameters**

device_info_hndl:
[Out] Pointer to device info handle.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

9.2.2. qapi_Device_Info_Get_v2

Gets the device information for specified ID.

- **Prototype**

```
qapi_Status_t qapi_Device_Info_Get_v2(qapi_Device_Info_Hndl_t device_info_hndl,
qapi_Device_Info_ID_t id, qapi_Device_Info_t *info);
```

- **Parameters**

device_info_hndl:
[In] Device info handle.

id:
[In] Information ID.

info:
[Out] Information received for the specified ID.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

Before calling this API, *qapi_Device_Info_Init_v2()* must have been called and a valid handle is obtained.

9.2.3. qapi_Device_Info_Set_Callback_v2

Sets a device information callback.

- **Prototype**

```
qapi_Status_t qapi_Device_Info_Set_Callback_v2(qapi_Device_Info_Hndl_t device_info_hndl,
qapi_Device_Info_ID_t id, qapi_Device_Info_Callback_t_v2 callback);
```

- **Parameters**

device_info_hndl:
[In] Device info handle.

id:
[In] Information ID.

callback:
[Out] Callback to be registered.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

Before calling this API, *qapi_Device_Info_Init_v2()* must have been called and a valid handle is obtained.

9.2.4. qapi_Device_Info_Release_v2

Releases the device information context.

- **Prototype**

```
qapi_Status_t qapi_Device_Info_Release_v2(qapi_Device_Info_Hndl_t device_info_hndl);
```

- **Parameters**

device_info_hndl:
[In] Device info handle.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

Before calling this API, *qapi_Device_Info_Init_v2()* must have been called and a valid handle is obtained.

9.2.5. qapi_Device_Info_Reset_v2

Resets the device.

- **Prototype**

```
qapi_Status_t qapi_Device_Info_Reset_v2(qapi_Device_Info_Hndl_t device_info_hndl);
```

- **Parameters**

device_info_hndl:
[In] Device info handle.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

Before calling this API, *qapi_Device_Info_Init_v2()* must have been called and a valid handle is obtained.

10 GPIO Interrupt Controller APIs

The general purpose input/output (GPIO) interrupt controller provides an interface for registering for interrupts for a GPIO. This can be done by configuring a GPIO as an input and toggling it externally to the chip. In doing so, this causes a GPIO interrupt to fire, and software will be invoked to handle it. Additionally, the register API will allow clients to register their callback, and the driver will internally configure the hardware to handle the given trigger type. Clients may also force-trigger the interrupt by using the trigger API, as well as check if an interrupt is pending by using the *Is_Interrupt_Pending( )* API. The GPIO interrupt may be enabled or disabled at any time using the Enable or Disable API. This ensures that the callback is not removed from the handler, but the interrupt will be unmasked/masked accordingly.

This chapter provides the following QAPIs:

```
qapi_GPIOINT_Register_Interrupt
qapi_GPIOINT_Deregister_Interrupt
qapi_GPIOINT_Set_Trigger
qapi_GPIOINT_Enable_Interrupt
qapi_GPIOINT_Disable_Interrupt
qapi_GPIOINT_Trigger_Interrupt
qapi_GPIOINT_Is_Interrupt_Pending
```

NOTE

Please refer to *Quectel_BG95&BG77_QuecOpen_Application_Note_V1.0* for pin definition of GPIO.

10.1. Data Types

10.1.1. Enumeration Type

10.1.1.1. enum qapi_GPIOINT_Trigger_e

This enumeration supports the GPIO interrupt trigger type for triggers.

```
typedef enum
{
QAPI_GPIOINT_TRIGGER_LEVEL_HIGH_E,
```

```
QAPI_GPIOINT_TRIGGER_LEVEL_LOW_E,
QAPI_GPIOINT_TRIGGER_EDGE_RISING_E,
QAPI_GPIOINT_TRIGGER_EDGE_FALLING_E,
QAPI_GPIOINT_TRIGGER_EDGE_DUAL_E,
} qapi_GPIOINT_Trigger_e;
```

- **Parameters**

Parameter	Description
QAPI_GPIOINT_TRIGGER_LEVEL_HIGH_E	Level triggered active high.
QAPI_GPIOINT_TRIGGER_LEVEL_LOW_E	Level triggered active low.
QAPI_GPIOINT_TRIGGER_EDGE_RISING_E	Rising edge triggered.
QAPI_GPIOINT_TRIGGER_EDGE_FALLING_E	Falling edge triggered.
QAPI_GPIOINT_TRIGGER_EDGE_DUAL_E	Dual edge triggered.

10.1.1.2. enum qapi_GPIOINT_Priority_e

This enumeration is used for GPIO interrupt priority selection.

```
typedef enum
{
    QAPI_GPIOINT_PRIO_HIGHEST_E,
    QAPI_GPIOINT_PRIO_HIGH_E,
    QAPI_GPIOINT_PRIO_MEDIUM_E,
    QAPI_GPIOINT_PRIO_LOW_E,
    QAPI_GPIOINT_PRIO_LOWEST_E,
} qapi_GPIOINT_Priority_e;
```

- **Parameters**

Parameter	Description
QAPI_GPIOINT_PRIO_HIGHEST_E	Highest priority.
QAPI_GPIOINT_PRIO_HIGH_E	Medium-high priority.
QAPI_GPIOINT_PRIO_MEDIUM_E	Medium priority.
QAPI_GPIOINT_PRIO_LOW_E	Medium-low priority.

QAPI_GPIOINT_PRIO_LOWEST_E	Lowest priority.

10.1.2. Typedefs

10.1.2.1. typedef uint32_t qapi_GPIOINT_Callback_Data_t

This is the data type of the argument passed into the callback that is registered with the GPIO interrupt module.

10.1.2.2. typedef void (* qapi_GPIOINT_CB_t)(qapi_GPIOINT_Callback_Data_t)

This function pointer is used for GPIO interrupt callback function definition.

10.1.2.3. typedef void* qapi_Instance_Handle_t

This pointer is used for GPIO interrupt handle definition.

10.2. API Functions

10.2.1. qapi_GPIOINT_Register_Interrupt

This function is used to register a callback for a GPIO interrupt. Register a callback function with the GPIO interrupt controller and enable the interrupt. This function configures and routes the interrupt accordingly, as well as enabling it in the underlying layers.

- **Prototype**

```
qapi_Status_t qapi_GPIOINT_Register_Interrupt(qapi_Instance_Handle_t *pH, uint32_t nGpio, qapi_GPIOINT_CB_t pfnCallback, qapi_GPIOINT_Callback_Data_t nData, qapi_GPIOINT_Trigger_e eTrigger, qapi_GPIOINT_Priority_e ePriority, qbool_t bNmi);
```

- **Parameters**

pH:
[In] Input handle to the client context.

nGpio:
[In] GPIO number to configure for an interrupt.

pfnCallback:

[In] Callback function pointer.

nData:
[In] Callback data.

eTrigger:
[In] Trigger type for the interrupt.

ePriority:
[In] Priority enumeration to determine the configuration of the GPIO interrupt.

bNmi:
[In] Boolean value to select whether or not the GPIO interrupt is nonmaskable to the CPU.

- **Return Value**

QAPI_ERR_INVALID_PARAM	There is an issue with one of the input parameters.
QAPI_ERROR	Error in internal registration.
QAPI_OK	Successfully registered.

- **Dependencies**

None.

NOTE

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

10.2.2. qapi_GPIOINT_Deregister_Interrupt

This function is used to undo the callback function from the GPIO interrupt controller and disable interrupts.

- **Prototype**

```
qapi_Status_t qapi_GPIOINT_Deregister_Interrupt ( qapi_Instance_Handle_t *pH, uint32_t nGpio);
```

- **Parameters**

pH:
[In] Input handle to the client context.

nGpio:
[In] GPIO number to deconfigure.

- **Return Value**

QAPI_ERROR	Error in internal deregistration.
QAPI_OK	Successfully deregistered

- **Dependencies**

None.

NOTE

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

10.2.3. qapi_GPIOINT_Set_Trigger

This function configures the underlying layer to capture an interrupt with a given trigger type. This function is only to be used on a currently registered GPIO interrupt and will change the trigger at runtime.

- **Prototype**

```
qapi_Status_t qapi_GPIOINT_Set_Trigger (qapi_Instance_Handle_t *pH, uint32_t nGpio,
qapi_GPIOINT_Trigger_e Trigger);
```

- **Parameters**

pH:
[In] Input handle to the client context.

nGpio:
[in] GPIO number in which to set the trigger.

eTrigger:
[In] Trigger type for configuration.

- **Return Value**

QAPI_ERR_INVALID_PARAM	*eTrigger* parameter is invalid.
QAPI_ERROR	Internal error in setting trigger.
QAPI_OK	Successfully set the trigger.

- **Dependencies**

None.

NOTE

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

10.2.4. qapi_GPIOINT_Enable_Interrupt

This function is used to enable a currently disabled and registered GPIO interrupt. This is used primarily to unmask interrupts.

- **Prototype**

```
qapi_Status_t qapi_GPIOINT_Enable_Interrupt ( qapi_Instance_Handle_t *pH, uint32_t nGpio);
```

- **Parameters**

pH:
[In] Input handle to the client context.

nGpio:
[In] GPIO number to enable.

- **Return Value**

QAPI_ERROR	Internal error in enabling interrupt.
QAPI_OK	Successfully enable interrupt.

- **Dependencies**

None.

NOTE

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

10.2.5. qapi_GPIOINT_Disable_Interrupt

This function is used to disable a currently enabled and registered GPIO interrupt. This is used primarily to mask interrupts, still being able to capture them, but not have the callback called.

- **Prototype**

```
qapi_Status_t qapi_GPIOINT_Disable_Interrupt (qapi_Instance_Handle_t *pH, uint32_t nGpio);
```

- **Parameters**

pH:
[In] Input handle to the client context.

nGpio:
[In] GPIO number to disable.

- **Return Value**

QAPI_ERROR	Internal error in disabling interrupt.
QAPI_OK	Successfully disabled interrupt.

- **Dependencies**

None.

NOTE

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

10.2.6. qapi_GPIOINT_Trigger_Interrupt

This function is used to manually trigger a GPIO interrupt by writing to the appropriate register.

- **Prototype**

```
qapi_Status_t qapi_GPIOINT_Trigger_Interrupt ( qapi_Instance_Handle_t *pH, uint32_t nGpio);
```

- **Parameters**

pH:
[In] Input handle to the client context.

nGpio:
[In] GPIO number to trigger.

- **Return Value**

QAPI_ERROR	Internal error in triggering interrupt.
QAPI_OK	Successfully triggered interrupt.

- **Dependencies**

None.

NOTE

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

10.2.7. qapi_GPIOINT_Is_Interrupt_Pending

This is used to queries to see if an interrupt is pending in the hardware by reading the appropriate register.

- **Prototype**

```
qapi_Status_t qapi_GPIOINT_Is_Interrupt_Pending (qapi_Instance_Handle_t *pH, uint32_t nGpio,
qbool_t *pbIsPending);
```

- **Parameters**

pH:
[In] Input handle to the client context.

nGpio:
[In] GPIO number to trigger.

pbIsPending:
[Out] Boolean value for whether or not the interrupt is pending in hardware.

- **Return Value**

QAPI_ERR_INVALID_PARAM	*pbIsPending* pointer is NULL.
QAPI_ERROR	Internal error in checking pending.
QAPI_OK	Successfully checked pending status.

- **Dependencies**

None.

NOTE

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

11 PMM APIs

Modern SoCs pack a lot of functionality but are often pin-limited owing to their shrinking size. This limitation is overcome by incorporating hardware to flexibly mux several different functionalities on a given physical pin under software control.

This module exposes an interface allowing its clients to manage desired functionalities on a set of physical GPIO pins on the SoC. The most common usage of this interface is to configure pins for discrete inputs or outputs to implement handshakes with external peripherals, sensors, or actuators.

This chapter provides the following QAPIs:

```
qapi_TLMM_Get_Gpio_ID
qapi_TLMM_Release_Gpio_ID
qapi_TLMM_Config_Gpio
qapi_TLMM_Drive_Gpio
qapi_TLMM_Read_Gpio
```

NOTE

Please refer to *Quectel_BG95&BG77_QuecOpen_Application_Note_V1.0* for pin definition of GPIO.

11.1. Data Types

11.1.1. Enumeration Type

11.1.1.1. enum qapi_GPIO_Direction_t

The enumeration is used to specify the direction when configuring a GPIO pin.

```
typedef enum
{
QAPI_GPIO_INPUT_E,
QAPI_GPIO_OUTPUT_E,
} qapi_GPIO_Direction_t;
```

- **Parameters**

Parameter	Description
QAPI_GPIO_INPUT_E	Specify the pin as an input to the SoC.
QAPI_GPIO_OUTPUT_E	Specify the pin as an output to the SoC.

11.1.1.2. enum qapi_GPIO_Pull_t

This enumeration specifies the type of pull to use when specifying the configuration for a GPIO pin.

```
typedef enum
{
QAPI_GPIO_NO_PULL_E,
QAPI_GPIO_PULL_DOWN_E,
QAPI_GPIO_KEEPER_E,
QAPI_GPIO_PULL_UP_E,
} qapi_GPIO_Pull_t;
```

- **Parameters**

Parameter	Description
QAPI_GPIO_NO_PULL_E	Specify no pull.
QAPI_GPIO_PULL_DOWN_E	Pull the GPIO down.
QAPI_GPIO_KEEPER_E	Keep the GPIO as it is.
QAPI_GPIO_PULL_UP_E	Pull the GPIO up.

11.1.1.3. enum qapi_GPIO_Drive_t

This enumeration specifies the drive strength to use when specifying the configuration of a GPIO pin.

```
typedef enum
{
QAPI_GPIO_2MA_E,
QAPI_GPIO_4MA_E,
QAPI_GPIO_6MA_E,
QAPI_GPIO_8MA_E,
```

```
QAPI_GPIO_10MA_E,
QAPI_GPIO_12MA_E,
QAPI_GPIO_14MA_E,
QAPI_GPIO_16MA_E,
} qapi_GPIO_Drive_t;
```

- **Parameters**

Parameter	Description
QAPI_GPIO_2MA_E	Specify a 2 mA drive.
QAPI_GPIO_4MA_E	Specify a 4 mA drive.
QAPI_GPIO_6MA_E	Specify a 6 mA drive.
QAPI_GPIO_8MA_E	Specify a 8 mA drive.
QAPI_GPIO_10MA_E	Specify a 10 mA drive.
QAPI_GPIO_12MA_E	Specify a 12 mA drive.
QAPI_GPIO_14MA_E	Specify a 14 mA drive.
QAPI_GPIO_16MA_E	Specify a 16 mA drive.

11.1.1.4. enum qapi_GPIO_Value_t

This enumeration specifies the value to write to a GPIO pin configured as an output.

```
typedef enum
{
QAPI_GPIO_LOW_VALUE_E,
QAPI_GPIO_HIGH_VALUE_E,
} qapi_GPIO_Value_t;
```

- **Parameters**

Parameter	Description
QAPI_GPIO_LOW_VALUE_E	Drive the output LOW.
QAPI_GPIO_HIGH_VALUE_E	Drive the output HIGH.

11.1.2. Typedefs

11.1.2.1. typedef uint16_t qapi_GPIO_ID_t

SoC pin access ID. The redefined data type used to receive the value of the obtained ID.

Unique ID will be provided by the module to the client. Clients must pass this ID as a token with subsequent calls. Please note that clients should cache the ID.

11.1.3. Structure Type

11.1.3.1. struct qapi_TLMM_Config_t

This structure is used to specify the configuration for a GPIO on the SoC.

```
typedef struct
{
uint32_t                pin;
uint32_t                func;
qapi_GPIO_Direction_t   dir;
qapi_GPIO_Pull_t        pull;
qapi_GPIO_Drive_t       drive;
} qapi_TLMM_Config_t;
```

- **Parameters**

Types	Parameter	Description
uint32_t	*pin*	Physical pin number.
uint32_t	*func*	Pin function selection.
qapi_GPIO_Direction_t	*dir*	Direction (input or output).
qapi_GPIO_Pull_t	*pull*	Pull value.
qapi_GPIO_Drive_t	*drive*	Drive strength.

11.2. API Functions

11.2.1. qapi_TLMM_Get_Gpio_ID

This function provides a unique access ID for a specified GPIO. This is required in order to access GPIO

Configuration APIs.

- **Prototype**

```
qapi_Status_t qapi_TLMM_Get_Gpio_ID (qapi_TLMM_Config_t *qapi_TLMM_Config,
qapi_GPIO_ID_t * qapi_GPIO_ID);
```

- **Parameters**

qapi_TLMM_Config:
[In] Pointer to the pin configuration data.

qapi_GPIO_ID:
[In] Pointer to a location in which to store the access ID.

- **Return Value**

QAPI_OK	Pin GPIO ID got successfully.
QAPI_ERR	Pin GPIO is currently in use or not programmable.

- **Dependencies**

None.

11.2.2. qapi_TLMM_Release_Gpio_ID

This function allows a client to relinquish the lock on a GPIO pin. It facilitates sharing of a pin between two drivers in different system modes where each driver may need to reconfigure the pin. This function do not need to be used unless such a condition dictates.

- **Prototype**

```
qapi_Status_t qapi_TLMM_Release_Gpio_ID(qapi_TLMM_Config_t *qapi_TLMM_Config,
qapi_GPIO_ID_t qapi_GPIO_ID);
```

- **Parameters**

qapi_TLMM_Config:
[In] Pointer to the pin configuration data.

qapi_GPIO_ID:
[In] Pin access ID retrieved from the *qapi_TLMM_Get_Gpio_ID()* call.

- **Return Value**

QAPI_OK	Pin was released successfully.
QAPI_ERR	Pin could not be released.

- **Dependencies**

None.

11.2.3. qapi_TLMM_Config_Gpio

This function configures an SoC pin based on a set of fields specified in the configuration structure reference passed in as a parameter.

- **Prototype**

```
qapi_Status_t qapi_TLMM_Config_Gpio(qapi_GPIO_ID_t qapi_GPIO_ID, qapi_TLMM_Config_t *
qapi_TLMM_Config);
```

- **Parameters**

qapi_GPIO_ID:
[In] Pin access ID retrieved from the *qapi_TLMM_Get_Gpio_ID()* call.

api_TLMM_Config:
[In] Pin configuration to be used.

- **Return Value**

QAPI_OK	Pin was configured successfully.
QAPI_ERR	Pin could not be configured.

- **Dependencies**

None.

11.2.4. qapi_TLMM_Drive_Gpio

This function drives the output of a SoC pin that has been configured as a generic output GPIO to a specified value.

- **Prototype**

```
qapi_Status_t qapi_TLMM_Drive_Gpio ( qapi_GPIO_ID_t qapi_GPIO_ID, uint32_t pin,
qapi_GPIO_Value_t value);
```

- **Parameters**

qapi_GPIO_ID:
[In] Pin access ID retrieved from the *qapi_TLMM_Get_Gpio_ID()* call.

pin:
[In] SoC pin number to configure.

value:
[In] Output value.

- **Return Value**

QAPI_OK	Operation completed successfully.
QAPI_ERR	Operation failed.

- **Dependencies**

None.

11.2.5. qapi_TLMM_Read_Gpio

This function is used to read the state of a SoC pin configured as an input GPIO.

- **Prototype**

```
qapi_Status_t qapi_TLMM_Read_Gpio (qapi_GPIO_ID_t qapi_GPIO_ID, uint32_t pin,
qapi_GPIO_Value_t *qapi_GPIO_Value);
```

- **Parameters**

qapi_GPIO_ID:
[In] Pin access ID retrieved from the *qapi_TLMM_Get_Gpio_ID()* call.

pin:
[In] SoC pin number to configure.

qapi_GPIO_Value:
[Out] GIPO pin value.

- **Return Value**

QAPI_GPIO_HIGH_VALUE	Read value was HIGH.
QAPI_GPIO_LOW_VALUE	Read value was LOW.

- **Dependencies**

None.

12 I2C APIs

I2C is a 2-wire bus used to connect low speed peripherals to a processor or a microcontroller. Common I2C peripherals include touch screen controllers, accelerometers, gyros, and ambient light and temperature sensors.

The 2-wire bus comprises a data line, a clock line, and basic START, STOP, and acknowledge signals to drive transfers on the bus. An I2C peripheral is also referred to as an I2C slave. The processor or microcontroller implements the I2C master as defined in the I2C specification. This documentation provides the software interface to access the I2C master implementation.

This chapter provides the following QAPIs:

```
qapi_I2CM_Open
qapi_I2CM_Close
qapi_I2CM_Transfer
qapi_I2CM_Power_On
qapi_I2CM_Power_Off
```

NOTE

Please refer to *Quectel_BG95&BG77_QuecOpen_Application_Note_V1.0* for pin definition of I2C.

12.1. Data Types

12.1.1. Enumeration Type

12.1.1.1. enum qapi_I2CM_Instance_t

This enumeration is the instance of the I2C core that the client wants to use.

```
typedef enum
{
QAPI_I2CM_INSTANCE_001_E = 1,
QAPI_I2CM_INSTANCE_002_E,
QAPI_I2CM_INSTANCE_003_E,
```

```
QAPI_I2CM_INSTANCE_004_E,
QAPI_I2CM_INSTANCE_005_E,
QAPI_I2CM_INSTANCE_006_E,
QAPI_I2CM_INSTANCE_007_E,
QAPI_I2CM_INSTANCE_008_E,
QAPI_I2CM_INSTANCE_009_E,
QAPI_I2CM_INSTANCE_010_E,
QAPI_I2CM_INSTANCE_011_E,
QAPI_I2CM_INSTANCE_012_E,
QAPI_I2CM_INSTANCE_013_E,
QAPI_I2CM_INSTANCE_014_E,
QAPI_I2CM_INSTANCE_015_E,
QAPI_I2CM_INSTANCE_016_E,
QAPI_I2CM_INSTANCE_017_E,
QAPI_I2CM_INSTANCE_018_E,
QAPI_I2CM_INSTANCE_019_E,
QAPI_I2CM_INSTANCE_020_E,
QAPI_I2CM_INSTANCE_021_E,
QAPI_I2CM_INSTANCE_022_E,
QAPI_I2CM_INSTANCE_023_E,
QAPI_I2CM_INSTANCE_024_E,
} qapi_I2CM_Instance_t;
```

NOTE

This enumeration is used in the macro definition, please refer to ***Chapter 12.1.2.2.***

12.1.2. Definition Type

12.1.2.1. I2C Transfer Status Macros

```
#define QAPI_I2C_FLAG_START          0x00000001
#define QAPI_I2C_FLAG_STOP           0x00000002
#define QAPI_I2C_FLAG_WRITE          0x00000004
#define QAPI_I2C_FLAG_READ           0x00000008
#define QAPI_I2C_TRANSFER_MASK    (QAPI_I2C_FLAG_WRITE|QAPI_I2C_FLAG_READ)
#define QAPI_VALID_FLAGS(x)      (((x & QAPI_I2C_TRANSFER_MASK) == QAPI_I2C_FLAG_READ)
|| ((x & QAPI_I2C_TRANSFER_MASK) ==QAPI_I2C_FLAG_WRITE))
```

- **Parameters**

Parameter	Description
QAPI_I2C_FLAG_START	Specifies that the transfer begins with a START bit – S.
QAPI_I2C_FLAG_STOP	Specifies that the transfer ends with a STOP bit – P.
QAPI_I2C_FLAG_WRITE	Must be set to indicate a WRITE transfer.
QAPI_I2C_FLAG_READ	Must be set to indicate a READ transfer.
QAPI_I2C_TRANSFER_MASK	Transfer types.
QAPI_VALID_FLAGS(x)	Verifies the validity of flags.

12.1.2.2. I2C Interface Definition

```
#define QT_QAPI_I2CM_PORT_01    QAPI_I2CM_INSTANCE_001_E
#define QT_QAPI_I2CM_PORT_02    QAPI_I2CM_INSTANCE_002_E
```

- **Parameters**

Parameter	Description
QT_QAPI_I2CM_PORT_01	I2C1 Port.
QT_QAPI_I2CM_PORT_02	I2C2 Port.

NOTE

The above three macros are associated with the enumeration *qapi_I2CM_Instance_t*, which are used to specify which port is to be opened during the *qapi_I2CM_Open()* call.

12.1.3. Typedefs

12.1.3.1. typedef void (*qapi_I2CM_Transfer_CB_t)(const uint32_t status,void *CB_Parameter)

This function pointer is used to declare the type of callback function that is to be defined by the client. The callback is called when the data is completely transferred on the bus or the transfer ends due to an error or cancellation. Clients pass the callback function pointer and the callback context to the driver in the *qapi_I2CM_Transfer()* API.

- **Prototype**

```
typedef void(*qapi_I2CM_Transfer_CB_t)(const uint32_t status,void *CB_Parameter);
```

- **Parameters**

status:
[Out] Completion status of the transfer. A call to *qapi_I2CM_Get_QStatus_Code()* will convert this status to QAPI status codes.

CB_Parameter:
[Out] CB_Parameter context that was passed in the call to *qapi_I2CM_Transfer()*.

- **Return Value**

None.

NOTE

The callback is called in the interrupt context. Calling any of the APIs defined here in the callback will result in the error *QAPI_I2CM_ERR_API_INVALID_EXECUTION_LEVEL*. Processing in the callback function must be kept to a minimum to avoid latencies in the system.

12.1.4. Structure Type

12.1.4.1. struct qapi_I2CM_Config_t

I2C client configuration parameters that the client uses to communicate to an I2C slave.

```
typedef struct
{
uint32_t     bus_Frequency_KHz;
uint32_t     slave_Address;
qbool_t     SMBUS_Mode;
uint32_t     slave_Max_Clock_Stretch_Us;
uint32_t     core_Configuration1;
uint32_t     core_Configuration2;
} qapi_I2CM_Config_t;
```

- **Parameters**

Types	Parameter	Description
uint32_t	*bus_Frequency_KHz*	I2C bus speed in kHz.

uint32_t	*slave_Address*	7-bit I2C slave address.
qbool_t	*SMBUS_Mode*	SMBUS mode transfers. Set to TRUE for SMBUS mode.
uint32_t	*slave_Max_Clock_Stretch_Us*	Maximum slave clock stretch in *slave_Max_Clock_Stretch_Us* that a slave might perform.
uint32_t	*core_Configuration1*	Core specific configuration, recommended to set it to 0.
uint32_t	*core_Configuration2*	Core specific configuration, recommended to set it to 0.

12.1.4.2. struct qapi_I2CM_Descriptor_t

I2C transfer descriptor

```
typedef struct
{
uint8_t      *buffer;
uint32_t     length;
uint32_t     transferred;
uint32_t     flags;
} qapi_I2CM_Descriptor_t;
```

- **Parameters**

Type	Parameter	Description
uint8_t	*buffer*	Buffer for the data transfer.
uint32_t	*length*	Length of the data to be transferred in bytes.
uint32_t	*transferred*	Number of bytes actually transferred.
uint32_t	*flags*	I2C flags for the transfer.

12.2. API Functions

12.2.1. qapi_I2CM_Open

The client calls this function to initialize the respective I2C instance. When this function is called successfully, *i2c_Handle* points to the handle for the I2C instance. The API allocates resources to be used by the client handle and the I2C instance. These resources are release in the *qapi_I2CM_Close()* call. The API also enables the power to the I2C HW instance. To disable the power to the instance, a corresponding call to *qapi_I2CM_Close()* must be made.

- **Prototype**

```
qapi_Status_t qapi_I2CM_Open (qapi_I2CM_Instance_t instance, void **c_Handle);
```

- **Parameters**

instance:
[In] I2C instance that the client intends to initialize; see *qapi_I2CM_Instance_t* for details.

i2c_Handle:
[Out] Pointer location to be filled by the driver with a handle to the instance.

- **Return Value**

QAPI_OK	Function was successful.
QAPI_I2CM_ERR_INVALID_PARAMETER	Invalid parameter.
QAPI_I2CM_ERR_API_INVALID_EXECUTION_LEVEL	Invalid execution level.
QAPI_I2CM_ERR_UNSUPPORTED_CORE_INSTANCE	Unsupported core instance.
QAPI_I2CM_ERR_HANDLE_ALLOCATION	Handle allocation error.
QAPI_I2CM_ERR_HW_INFO_ALLOCATION	Hardware information allocation error
QAPI_I2CM_ERR_PLATFORM_INIT_FAIL	Platform initialization failure.
QAPI_I2CM_ERR_PLATFORM_REG_INT_FAIL	Platform registration internal failure
QAPI_I2CM_ERR_PLATFORM_CLOCK_ENABLE_FAIL	Platform clock enable failure.
QAPI_I2CM_ERR_PLATFORM_GPIO_ENABLE_FAIL	Platform GPIO enable failure.

- **Dependencies**

None.

12.2.2. qapi_I2CM_Close

This function is used to close the I2C and releases any resources allocated by the *qapi_I2CM_Open()*.

- **Prototype**

```
qapi_Status_t qapi_I2CM_Close (void **i2c_Handle);
```

- **Parameters**

i2c_Handle:
[In] Handle to the I2C instance.

- **Return Value**

QAPI_OK	Function was successful.
QAPI_I2CM_ERR_INVALID_PARAMETER	Invalid parameter.
QAPI_I2CM_ERR_API_INVALID_EXECUTION_LEVEL	Invalid execution level.

QAPI_I2CM_ERR_PLATFORM_DEINIT_FAIL	Platform de-initialization failure.
QAPI_I2CM_ERR_PLATFORM_DE_REG_INT_FAIL	Platform de-registration internal failure.
QAPI_I2CM_ERR_PLATFORM_CLOCK_DISABLE_FAIL	Platform clock disabling failure.
QAPI_I2CM_ERR_PLATFORM_GPIO_DISABLE_FAIL	Platform GPIO disabling failure.

- **Dependencies**

None.

12.2.3. qapi_I2CM_Transfer

This function is used to perform an I2C transfer. In case a transfer is already in progress by another client, this call queues the transfer. If the transfer returns a failure, the transfer has not been queued and no callback will occur. If the transfer returns *QAPI_OK*, the transfer has been queued and a further status of the transfer can only be obtained when the callback is called.

- **Prototype**

```
qapi_Status_t qapi_I2CM_Transfer (void * i2c_Handle, qapi_I2CM_Config_t *config,
qapi_I2CM_Descriptor_t *desc, uint16_t num_Descriptors,qapi_I2CM_Transfer_CB_t CB_Function,
void * CB_Parameter, uint32_t delay_us);
```

- **Parameters**

i2c_Handle:
[In] Handle to the I2C instance.

config:
[In] Slave configuration. See *qapi_I2CM_Config_t* for details.

desc:
[In] I2C transfer descriptor. See *qapi_I2CM_Descriptor_t* for details. This can be an array of descriptors.

num_Descriptors:
[In] Number of descriptors in the descriptor array.

CB_Function:
[In] Callback function that is called at the completion of the transfer occurs in the interrupt context. The call must do minimal processing and must not call any API defined here.

CB_Parameter:
[In] Context that the client passes here is returned as is in the callback function.

delay_us:
[In] Delay in microseconds that specifies the time to wait before starting the transfer.

- **Return Value**

QAPI_OK	Function was successful.
QAPI_I2CM_ERR_INVALID_PARAMETER	Invalid parameter.
QAPI_I2CM_ERR_API_INVALID_EXECUTION_LEVEL	Invalid execution level.
QAPI_I2CM_ERR_TRANSFER_TIMEOUT	Transfer timed out.
QAPI_I2CM_ERR_QSTATE_INVALID	QState is invalid.
QAPI_I2CM_ERROR_HANDLE_ALREADY_IN_QUEUE	Client IO is pending.

- **Dependencies**

None.

NOTE

After a client calls this API, it must wait for the completing the callback to occur before another API can be called again. If the client wishes to queue mutliple transfers, it must use an array of descriptors of type *qapi_I2CM_Descriptor_t* instead of calling the API multiple times.

12.2.4. qapi_I2CM_Power_On

This function is used to enable the I2C hardware resources for an I2C transaction. This function enables all resources required for a successful I2C transaction. The involved resources include clocks, power resources and pin multiplex functions.

- **Prototype**

```
qapi_Status_t qapi_I2CM_Power_On(void *i2c_Handle);
```

- **Parameters**

i2c_Handle:
[in] Driver handle returned by *qapi_I2CM_Open()*.

- **Return Value**

QAPI_OK	I2C master enabled successfully.
QAPI_I2CM_ERROR_INVALID_PARAM	Invalid handle parameter.

- **Dependencies**

qapi_I2CM_Open() must have been called first.

12.2.5. qapi_I2CM_Power_Off

This function is used to disable the I2C hardware resources for an I2C transaction. This function turns off

all resources used by the I2C master. The involved resources include clocks, power resources and GPIOs. This function should be called to put the I2C master under its lowest possible power state.

- **Prototype**

```
qapi_Status_t qapi_I2CM_Power_Off (void *i2c_Handle);
```

- **Parameters**

i2c_Handle:
[In] Driver handle returned by *qapi_I2CM_Open()*.

- **Return Value**

QAPI_OK	I2C master disabled successfully.
QAPI_I2CM_ERROR_INVALID_PARAM	Invalid handle parameter.

- **Dependencies**

qapi_I2CM_Open() must have been called first.

13 SPI APIs

The serial peripheral interface (SPI) is a full duplex communication bus to interface peripherals in several communication modes as configured by the client software. The SPI driver API provides a high-level interface to expose the capabilities of the SPI master.

This chapter provides the following QAPIs:

```
qapi_SPIM_Open
qapi_SPIM_Power_On
qapi_SPIM_Power_Off
qapi_SPIM_Full_Duplex
qapi_SPIM_Close
```

NOTE

Please refer to *Quectel_BG95&BG77_QuecOpen_Application_Note_V1.0* for pin definition of SPI.

13.1. Data Types

13.1.1. Enumeration Type

13.1.1.1. enum qapi_SPIM_Instance_t

This enumeration lists the possible SPI instance indicating which HW SPI master is to be used for the current SPI transaction.

```
typedef enum
{
QAPI_SPIM_INSTANCE_1_E = 1,
QAPI_SPIM_INSTANCE_2_E,
QAPI_SPIM_INSTANCE_3_E,
QAPI_SPIM_INSTANCE_4_E,
QAPI_SPIM_INSTANCE_5_E,
QAPI_SPIM_INSTANCE_6_E,
QAPI_SPIM_INSTANCE_7_E,
```

```
QAPI_SPIM_INSTANCE_8_E,
QAPI_SPIM_INSTANCE_9_E,
QAPI_SPIM_INSTANCE_10_E,
QAPI_SPIM_INSTANCE_11_E,
QAPI_SPIM_INSTANCE_12_E,
QAPI_SPIM_INSTANCE_13_E,
QAPI_SPIM_INSTANCE_14_E,
QAPI_SPIM_INSTANCE_15_E,
QAPI_SPIM_INSTANCE_16_E,
QAPI_SPIM_INSTANCE_17_E,
QAPI_SPIM_INSTANCE_18_E,
QAPI_SPIM_INSTANCE_19_E,
QAPI_SPIM_INSTANCE_20_E,
QAPI_SPIM_INSTANCE_21_E,
QAPI_SPIM_INSTANCE_22_E,
QAPI_SPIM_INSTANCE_23_E,
QAPI_SPIM_INSTANCE_24_E,
} qapi_SPIM_Instance_t;
```

NOTE

This enumeration is used in the macro definition, please refer to ***Chapter 13.1.2.1.***

13.1.1.2. enum qapi_SPIM_Shift_Mode_t

This type defines the clock phase that the client can set in the SPI configuration.

```
typedef enum
{
QAPI_SPIM_MODE_0_E,
QAPI_SPIM_MODE_1_E,
QAPI_SPIM_MODE_2_E,
QAPI_SPIM_MODE_3_E,
} qapi_SPIM_Shift_Mode_t;
```

- **Parameters**

Parameter	Description
QAPI_SPIM_MODE_0_E	CPOL=0, CPHA=0.
QAPI_SPIM_MODE_1_E	CPOL=0, CPHA=1.

QAPI_SPIM_MODE_2_E	CPOL=1, CPHA=0.
QAPI_SPIM_MODE_3_E	CPOL=1, CPHA=1.

13.1.1.3. enum qapi_SPIM_CS_Polarity_t

SPI chip selection polarity type.

```
typedef enum
{
QAPI_SPIM_CS_ACTIVE_LOW_E,
QAPI_SPIM_CS_ACTIVE_HIGH_E,
} qapi_SPIM_CS_Polarity_t;
```

- **Parameters**

Parameter	Description
QAPI_SPIM_CS_ACTIVE_LOW_E	During Idle state, the CS line is held low.
QAPI_SPIM_CS_ACTIVE_HIGH_E	During Idle state, the CS line is held high.

13.1.1.4. enum qapi_SPIM_Byte_Order_t

Order in which bytes from TX/RX buffer words are put on the bus.

```
typedef enum
{
SPI_NATIVE = 0,
SPI_LITTLE_ENDIAN = 0,
SPI_BIG_ENDIAN
} qapi_SPIM_Byte_Order_t;
```

- **Parameters**

Parameter	Description
SPI_NATIVE	Native
SPI_LITTLE_ENDIAN	Little Endian
SPI_BIG_ENDIAN	Big Endian (network)

13.1.1.5. enum qapi_SPIM_CS_Mode_t

This type defines how the chip selection line is configured between N word cycles.

```
typedef enum
{
QAPI_SPIM_CS_DEASSERT_E,
QAPI_SPIM_CS_KEEP_ASSERTED_E,
} qapi_SPIM_CS_Mode_t;
```

- **Parameters**

Parameter	Description
QAPI_SPIM_CS_DEASSERT_E	CS is de-asserted after transferring data for N clock cycles.
QAPI_SPIM_CS_KEEP_ASSERTED_E	CS is asserted as long as the core is in the Run state.

13.1.2. Definition Type

13.1.2.1. SPI Interface Definition

```
#define QT_QAPI_SPIM_PORT_01    QAPI_SPIM_INSTANCE_1_E
#define QT_QAPI_SPIM_PORT_02    QAPI_SPIM_INSTANCE_2_E
#define QT_QAPI_SPIM_PORT_03    QAPI_SPIM_INSTANCE_4_E
```

- **Parameters**

Parameter	Description
QT_QAPI_SPIM_PORT_01	SPI1 Port.
QT_QAPI_SPIM_PORT_02	SPI2 Port.
QT_QAPI_SPIM_PORT_03	SPI3 Port.

NOTE

The above three macros are associated with the enumeration *qapi_SPIM_Instance_t* which are used to specify which port is to be opened during the *qapi_SPIM_Open call*.

13.1.3. Typedefs

13.1.3.1. typedef void (* qapi_SPIM_Callback_Fn_t)(uint32_t status, void *callback_Ctxt)

This type is used by the client to register its callback notification function. The *callback_Ctxt* is the context object that will be passed untouched by the SPI Master driver to help the client identify which transfer completion instance is being signalled.

- **Prototype**

```
typedef void(*qapi_SPIM_Callback_Fn_t)(uint32_t status, void *callback_Ctxt);
```

- **Parameters**

status:
[Out] Completion status of the transfer. A call to *qapi_SPIM_Get_QStatus_Code()* will convert this status to QAPI status codes.

callback_Ctxt:
[Out] callback_Ctxt context that was passed in the call to *qapi_SPIM_Full_Duplex()*.

- **Return Value**

None.

13.1.4. Structure Type

13.1.4.1. struct qapi_SPIM_Config_t

The SPI master configuration is the collection of settings specified for each SPI transfer call to select the various possible SPI transfer parameters.

```
typedef struct
{
qapi_SPIM_Shift_Mode_t SPIM_Mode;
qapi_SPIM_CS_Polarity_t SPIM_CS_Polarity;
qapi_SPIM_Byte_Order_t SPIM_endianness;
uint8_t SPIM_Bits_Per_Word;
uint8_t SPIM_Slave_Index;
uint32_t Clk_Freq_Hz;
uint32_t CS_Clk_Delay_Cycles; //QuectelModifyFlag
uint32_t Inter_Word_Delay_Cycles; //QuectelModifyFlag
qapi_SPIM_CS_Mode_t SPIM_CS_Mode;
qbool_t loopback_Mode;
```

```
} qapi_SPIM_Config_t;
```

- **Parameters**

Type	Parameter	Description
qapi_SPIM_Shift_Mode_t	*SPIM_Mode*	SPIM mode type to be used for the SPI core.
qapi_SPIM_CS_Polarity_t	*SPIM_CS_Polarity*	CS polarity type to be used for the SPI core.
qapi_SPIM_Byte_Order_t	*SPIM_endianness*	Endian-ness type used for the SPI transfer.
uint8_t	*SPIM_Bits_Per_Word*	Endian-ness type used for the SPI transfer. SPI bits per word; any value from 3 to 31.
uint8_t	*SPIM_Slave_Index*	Slave index, beginning at 0 if multiple SPI devices are connected to the same master. At most 7 slaves are allowed. If an invalid number (7 or higher) is set, the CS signal will not be used.
uint32_t	*Clk_Freq_Hz*	Host sets the SPI clock frequency closest to the requested frequency.
uint32_t	*CS_Clk_Delay_Cycles*	Number of clock cycles to wait after asserting CS before starting transfer.
uint32_t	*Inter_Word_Delay_Cycles*	Number of clock cycles to wait between SPI words.
qapi_SPIM_CS_Mode_t	*SPIM_CS_Mode*	CS mode to be used for the SPI core.
qbool_t	*loopback_Mode*	In general, the value is 0. If set, the SPI controller will enable Loopback mode; used primarily for testing.

13.1.4.2. struct qapi_SPIM_Descriptor_t

This type specifies the address and length of the buffer for an SPI transaction.

```
typedef struct
{
uint8_t     *tx_buf;
uint8_t     *rx_buf;
uint32_t     len;
} qapi_SPIM_Descriptor_t;
```

- **Parameters**

Types	Parameter	Description
uint8_t *	*tx_buf*	Buffer address for transmitting data.
uint8_t *	*rx_buf*	Buffer address for receiving data.
uint32_t	*len*	Size in bytes. No alignment requirements; the arbitrary length data can be transferred.

13.2. API Functions

13.2.1. qapi_SPIM_Open

This function is used to initialize internal data structures along with associated static data. In any operating mode, this function should be called before calling any other SPI master API.

- **Prototype**

```
qapi_Status_t qapi_SPIM_Open (qapi_SPIM_Instance_t instance, void **spi_Handle);
```

- **Parameters**

instance:
[In] SPI instance specified by *qapi_SPIM_Instance_t*.

spi_Handle:
[Out] Pointer to a location in which to store the driver handle.

- **Return Value**

QAPI_OK	Module initialized successfully.
QAPI_SPIM_ERROR_INVALID_PARAM	Invalid instance or handle parameter.
QAPI_SPIM_ERROR_MEM_ALLOC	Could not allocate space for driver structures.

- **Dependencies**

None.

13.2.2. qapi_SPIM_Power_On

This function is used to enable the SPI hardware resources for an SPI transaction, these resources include clocks, power resources and pin multiplex functions. This function should be called before a transfer or a batch of SPI transfers.

- **Prototype**

```
qapi_Status_t qapi_SPIM_Power_On(void * spi_Handle);
```

- **Parameters**

spi_Handle:
[In] Driver handle returned by *qapi_SPIM_Open()*.

- **Return Value**

QAPI_OK	SPI master enabled successfully.
QAPI_SPIM_ERROR_INVALID_PARAM	Invalid handle parameter.

- **Dependencies**

qapi_SPIM_Open() must have been called first.

13.2.3. qapi_SPIM_Power_Off

This function is used to disable the SPI hardware resources for an SPI transaction.

- **Prototype**

```
qapi_Status_t qapi_SPIM_Power_Off(void * spi_Handle);
```

- **Parameters**

spi_Handle:
[In] Driver handle returned by *qapi_SPIM_Open()*.

- **Return Value**

QAPI_OK	SPI master disabled successfully.
QAPI_SPIM_ERROR_INVALID_PARAM	Invalid handle parameter.

- **Dependencies**

None.

13.2.4. qapi_SPIM_Full_Duplex

This function is used to perform an asynchronous transfer over the SPI bus. Transfers can be one-directional or bi-directional. A callback is generated upon transfer completion.

- **Prototype**

```
qapi_Status_t qapi_SPIM_Full_Duplex(void * spi_Handle, qapi_SPIM_Config_t * config, qapi_SPIM_Descriptor_t *desc, uint32_t num_Descriptors, qapi_SPIM_Callback_Fn_t c_Fn, void *c_Ctxt, qbool_t get_timestamp);
```

- **Parameters**

spi_Handle:
[In] Driver handle returned by *qapi_SPIM_Open()*.

config:
[In] Pointer to the SPI configuration structure described by *qapi_SPIM_Config_t*.

desc:
[In] Pointer to the structure described by *qapi_SPIM_Descriptor_t*. The descriptor can have NULL TX or RX buffers if a half-duplex transfer is selected.

num_Descriptors:
[In] Number of descriptor pointers that the client wants to process.

c_Fn:
[In] Callback function to be invoked when the SPI transfer completed successfully or with an error.

c_Ctxt:
[In] Pointer to a client object that will be returned as an argument to *c_Fn*.

get_timestamp:
[In] Boolean variable to indicate whether a transaction timestamp needs to be provided; this is not supported for the QUPv2 version.

- **Return Value**

QAPI_OK	SPI master enabled successfully.
QAPI_SPIM_ERROR_INVALID_PARAM	One or more invalid parameters.

- **Dependencies**

None.

13.2.5. qapi_SPIM_Close

This function is used to free all internal data structures and close SPI master interface.

- **Prototype**

```
qapi_Status_t qapi_SPIM_Close(void *spi_handle);
```

- **parameters**

spi_Handle:
[In] Driver handle returned by *qapi_SPIM_Open()*.

- **Return Value**

QAPI_OK	SPI driver closed successfully.
QAPI_SPIM_ERROR_INVALID_PARAM	One or more invalid parameters.
QAPI_SPIM_ERROR_CLOSE_FAILURE	SPI driver closed failure.

- **Dependencies**

None.

14 UART APIs

This section describes the UART data types and APIs.

This chapter provides the following QAPIs:

```
qapi_UART_Open
qapi_UART_Close
qapi_UART_Receive
qapi_UART_Transmit
qapi_UART_Power_On
qapi_UART_Power_Off
qapi_UART_Ioctl
```

NOTE

Please refer to *Quectel_BG95&BG77_QuecOpen_Application_Note_V1.0* for pin definition of UART.

14.1. Data Types

14.1.1. Enumeration Type

14.1.1.1. enum qapi_UART_Port_Id_e

This enumeration is used to specify which port is to be opened.

```
typedef enum
{
    QAPI_UART_PORT_001_E,
    QAPI_UART_PORT_002_E,
    QAPI_UART_PORT_003_E,
    QAPI_UART_PORT_004_E,
    QAPI_UART_PORT_005_E,
    QAPI_UART_PORT_006_E,
    QAPI_UART_PORT_007_E,
    QAPI_UART_PORT_008_E,
```

```
    QAPI_UART_PORT_009_E,
    QAPI_UART_PORT_010_E,
    QAPI_UART_PORT_011_E,
    QAPI_UART_PORT_012_E,
    QAPI_UART_PORT_013_E,
    QAPI_UART_PORT_014_E,
    QAPI_UART_PORT_015_E,
    QAPI_UART_PORT_016_E,
    QAPI_UART_PORT_017_E,
    QAPI_UART_PORT_018_E,
    QAPI_UART_PORT_019_E,
    QAPI_UART_PORT_020_E,
    QAPI_UART_PORT_021_E,
    QAPI_UART_PORT_022_E,
    QAPI_UART_PORT_023_E,
    QAPI_UART_PORT_024_E,
} qapi_UART_Port_Id_e;
```

NOTE

This enumeration is used in the macro definition, please refer to ***Chapter 14.1.2.1.***

14.1.1.2. enum qapi_UART_Bits_Per_Char_e

Enumeration to specify how many UART bits are to be used per character configuration.

```
typedef enum
{
QAPI_UART_5_BITS_PER_CHAR_E,
QAPI_UART_6_BITS_PER_CHAR_E,
QAPI_UART_7_BITS_PER_CHAR_E,
QAPI_UART_8_BITS_PER_CHAR_E,
} qapi_UART_Bits_Per_Char_e;
```

- **Parameters**

Parameter	Description
QAPI_UART_5_BITS_PER_CHAR_E	5 bits per character.
QAPI_UART_6_BITS_PER_CHAR_E	6 bits per character.

QAPI_UART_7_BITS_PER_CHAR_E	7 bits per character.
QAPI_UART_8_BITS_PER_CHAR_E	8 bits per character.

14.1.1.3. enum qapi_UART_Num_Stop_Bits_e

Enumeration for UART number of stop bits configuration.

```
typedef enum
{
QAPI_UART_0_5_STOP_BITS_E,
QAPI_UART_1_0_STOP_BITS_E,
QAPI_UART_1_5_STOP_BITS_E,
QAPI_UART_2_0_STOP_BITS_E,
} qapi_UART_Num_Stop_Bits_e;
```

- **Parameters**

Parameter	Description
QAPI_UART_0_5_STOP_BITS_E	0.5 stop bits.
QAPI_UART_1_0_STOP_BITS_E	1.0 stop bits.
QAPI_UART_1_5_STOP_BITS_E	1.5 stop bits.
QAPI_UART_2_0_STOP_BITS_E	2.0 stop bits.

14.1.1.4. enum qapi_UART_Parity_Mode_e

Enumeration for UART parity mode configuration.

```
typedef enum
{
QAPI_UART_NO_PARITY_E,
QAPI_UART_ODD_PARITY_E,
QAPI_UART_EVEN_PARITY_E,
QAPI_UART_SPACE_PARITY_E,
} qapi_UART_Parity_Mode_e;
```

- **Parameters**

Parameter	Description
QAPI_UART_NO_PARITY_E	No parity.
QAPI_UART_ODD_PARITY_E	Odd parity.
QAPI_UART_EVEN_PARITY_E	Even parity.
QAPI_UART_SPACE_PARITY_E	Space parity.

14.1.1.5. enum qapi_UART_Ioctl_Command_e

IOCTL command ID list of the UART.

```
typedef enum
{
QAPI_SET_FLOW_CTRL_E,
QAPI_SET_BAUD_RATE_E,
} qapi_UART_Ioctl_Command_e;
```

- **Parameters**

Parameter	Description
QAPI_SET_FLOW_CTRL_E	Set auto flow control.
QAPI_SET_BAUD_RATE_E	Set baud rate.

14.1.1.6. enum QAPI_Flow_Control_Type

Flow control types for UART.

```
typedef enum
{
QAPI_FCTL_OFF_E,
QAPI_CTSRFR_AUTO_FCTL_E
} QAPI_Flow_Control_Type;
```

- **Parameters**

Parameter	Description
QAPI_FCTL_OFF_E	Disable flow control.
QAPI_CTSRFR_AUTO_FCTL_E	Use CTS/RFR flow control with auto RX RFR signal generation.

14.1.2. Definition Type

14.1.2.1. UART Interface Definition

```
#define QT_QAPI_UART_PORT_01    QAPI_UART_PORT_001_E
#define QT_QAPI_UART_PORT_02    QAPI_UART_PORT_002_E
#define QT_QAPI_UART_PORT_03    QAPI_UART_PORT_003_E
```

- **Parameters**

Parameter	Description
QT_QAPI_UART_PORT_01	UART1
QT_QAPI_UART_PORT_02	UART2
QT_QAPI_UART_PORT_03	UART3

NOTE

The above three macros are associated with the enumeration *qapi_UART_Port_Id_e*, which are used to specify which port is to be opened during the *qapi_UART_Open( )* call.

14.1.3. Typedefs

14.1.3.1. typedef void* qapi_UART_Handle_t

UART handle that is passed to the client when a UART port is opened.

14.1.3.2. typedef void (*qapi_UART_Callback_Fn_t)(uint32_t num_bytes, void *cb_data)

Transmit and Receive operation callback type.

- **Prototype**

```
typedef void(*qapi_UART_Callback_Fn_t)(uint32_t num_bytes, void *cb_data);
```

- **Parameters**

num_bytes:
[In] Number of bytes.

cb_data:
[Out] Pointer to the callback data.

- **Return Value**

None.

14.1.4. Structure Type

14.1.4.1. union QAPI_UART_Ioctl_Param

IOCTL command ID list of the UART.

```
typedef union
{
uint32_t                baud_Rate;
QAPI_Flow_Control_Type  Flow_Control_Type;
} QAPI_UART_Ioctl_Param;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*baud_Rate*	Supported baud rates are 115200 bps, 1 Mbps,2 Mbps, 3 Mbps, and 4 Mbps.
QAPI_Flow_Control_Type	*Flow_Control_Type*	Transmit flow control type.

14.1.4.2. struct qapi_UART_Open_Config_t

Structure for UART configuration.

```
typedef struct
{
```

```
uint32_t                    baud_Rate;
qapi_UART_Parity_Mode_e     parity_Mode;
qapi_UART_Num_Stop_Bits_e   num_Stop_Bits;
qapi_UART_Bits_Per_Char_e   bits_Per_Char;
qbool_t                     enable_Loopback;
qbool_t                     enable_Flow_Ctrl;
boolean                     user_mode_client;
qapi_UART_Callback_Fn_t     tx_CB_ISR;
qapi_UART_Callback_Fn_t     rx_CB_ISR;
} qapi_UART_Open_Config_t;
```

- **Parameters**

Type	Parameter	Description
uint32_t	*baud_Rate*	Supported baud rates are 115200 bps, 1 Mbps,2 Mbps, 3 Mbps, and 4 Mbps.
qapi_UART_Parity_Mode_e	*parity_Mode*	Parity mode.
qapi_UART_Num_Stop_Bits_e	*num_Stop_Bits*	Number of stop bits.
qapi_UART_Bits_Per_Char_e	*bits_Per_Char*	Bits per character.
qbool_t	*enable_Loopback*	Enable loopback.
qbool_t	*enable_Flow_Ctrl*	Enable flow control.
boolean	*user_mode_client*	Reserve for QAPI UART Driver.
qapi_UART_Callback_Fn_t	*tx_CB_ISR*	Transmit callback, called from ISR context.
qapi_UART_Callback_Fn_t	*rx_CB_ISR*	Receive callback, called from ISR context.

14.2. API Functions

14.2.1. qapi_UART_Open

This function is used to open the UART port and configure the corresponding clocks, interrupts and GPIO.

- **Prototype**

```
qapi_Status_t qapi_UART_Open(qapi_UART_Handle_t *handle, qapi_UART_Port_Id_e id, qapi_UART_Open_Config_t *config);
```

- **Parameters**

handle:
[Out] UART handle.

id:
[In] ID of the port to be opened.

config:
[In] Structure that holds all configuration data.

- **Return Value**

QAPI_OK	Port open was successful.
QAPI_ERROR	Port open failed.

- **Dependencies**

None.

NOTE

Please do not call this API from ISR context.

14.2.2. qapi_UART_Close

This function is used to release clock, interrupt, and GPIO handles related to this UART and cancel any pending transfers.

- **Prototype**

```
qapi_Status_t qapi_UART_Close(qapi_UART_Handle_t handle);
```

- **Parameters**

handle:
[In] UART handle provided by *qapi_UART_Open()*.

- **Return Value**

QAPI_OK	Port close was successful.
QAPI_ERROR	Port close failed.

- **Dependencies**

None.

NOTE

Do not call this API from ISR context.

14.2.3. qapi_UART_Receive

This function is used to queue the buffer provided for receiving the data. This is an asynchronous call. *rx_cb_isr* is called when RX transfer completion. The buffer is owned by the UART driver until *rx_cb_isr* is called. There must be always a pending RX. The UART hardware has a limited buffer (FIFO), and if there is no software buffer available for HS-UART, the flow control will de-assert the RFR line. Call *uart_receive* immediately after *uart_open* to queue a buffer. After every *rx_cb_isr*, from a different non-ISR thread, queue the next transfer. There can be two buffers queued at a time at most.

- **Prototype**

```
qapi_Status_t qapi_UART_Receive(qapi_UART_Handle_t handle, char *buf, uint32_t buf_Size, void*cb_Data);
```

- **Parameters**

handle:
[In] UART handle provided by *qapi_UART_Open()*.

buf:
[Out] Buffer to be filled with data.

buf_Size:
[In] Buffer size. Must be ≥ 4 and a multiple of 4.

cb_Data:
[In] Callback data to be passed when *rx_cb_isr* is called during RX completion.

- **Return Value**

QAPI_OK	Queuing of the receive buffer was successful.
QAPI_ERROR	Queuing of the receive buffer failed.

- **Dependencies**

None.

NOTE

Please do not call this API from ISR context.

14.2.4. qapi_UART_Transmit

This function is used to transmit data from a specified buffer.

- **Prototype**

```
qapi_Status_t qapi_UART_Transmit(qapi_UART_Handle_t handle, char *buf, uint32_t bytes_To_Tx, void *cb_Data);
```

- **Parameters**

handle:
[In] UART handle provided by *qapi_UART_Open()*.

buf:
[In] Buffer with data for transmit.

bytes_To_Tx:
[In] Bytes of data to transmit.

cb_Data:
[In] Callback data to be passed when *tx_cb_isr* is called during TX completion.

- **Return Value**

QAPI_OK	Queuing of the transmit buffer was successful.
QAPI_ERROR	Queuing of the transmit buffer failed.

- **Dependencies**

None.

NOTE

Please do not call this API from ISR context.

14.2.5. qapi_UART_Power_On

This function is used to enable the UART hardware resources for a UART transaction, these resources include clocks, power resources and pin multiplex functions. This function should be called before a transfer or a batch of UART transfers.

- **Prototype**

```
qapi_Status_t qapi_UART_Power_On(qapi_UART_Handle_t UART_Handle);
```

- **Parameters**

UART_Handle:
[In] Driver handle returned by *qapi_UART_Open()*.

- **Return Value**

QAPI_OK	UART powered on successfully.
QAPI_ERROR	UART power on failed.

- **Dependencies**

qapi_UART_Open() must have been called first.

14.2.6. qapi_UART_Power_Off

This function is used to disable the UART hardware resources for a UART transaction, these resources include clocks, power resources, and GPIOs. This function should be called to put the UART master in its lowest possible power state.

- **Prototype**

```
qapi_Status_t qapi_UART_Power_Off(qapi_UART_Handle_t UART_Handle);
```

- **Parameters**

UART_Handle:
[In] Driver handle returned by *qapi_UART_Open()*.

- **Return Value**

QAPI_OK	UART powered off successfully.
QAPI_ERROR	UART power off failed.

- **Dependencies**

None.

14.2.7. qapi_UART_Ioctl

This function is used to controls the UART configuration for a UART transaction

- **Prototype**

```
qapi_Status_t qapi_UART_Ioctl(qapi_UART_Handle_t handle, qapi_UART_Ioctl_Command_e
ioctl_Command, void *ioctl_Param);
```

- **Parameters**

handle:
[In] Driver handle returned by *qapi_UART_Open()*.

ioctl_Command:
[In] Pass the commands listed with *qapi_UART_Ioctl_Command_e*.

ioctl_Param:
[In] Pass the argument associated with *qapi_UART_Ioctl_Command_e*.

- **Return Value**

QAPI_OK	UART configured successfully.
QAPI_ERROR	UART configuration failed.

- **Dependencies**

None.

15 Timer APIs

This chapter provides the following APIs for timer services. This timer service is different than the RTOS timer service.

```
qapi_Timer_Def
qapi_Timer_Set
qapi_Timer_Get_Timer_Info
qapi_Timer_Sleep
qapi_Timer_Undef
qapi_Timer_Stop
```

15.1. Data Types

15.1.1. Typedef Type

15.1.1.1. enum qapi_TIMER_notify_t

Enumeration of the notifications available on timer expiry.

```
typedef enum
{
   QAPI_TIMER_NO_NOTIFY_TYPE    = 0x00,
   QAPI_TIMER_NATIVE_OS_SIGNAL_TYPE,
   QAPI_TIMER_FUNC1_CB_TYPE
} qapi_TIMER_notify_t;
```

- **Parameters**

Parameter	Description
QAPI_TIMER_NO_NOTIFY_TYPE	No notification.
QAPI_TIMER_NATIVE_OS_SIGNAL_TYPE	Signal an object.
QAPI_TIMER_FUNC1_CB_TYPE	Call back a function.

15.1.1.2. typedef void* qapi_TIMER_handle_t

Timer handle.

Handle provided by the timer module to the client. Clients must pass this handle as a token with subsequent timer calls. Note that the clients should cache the handle. Once lost, it cannot be queried back from the module.

15.1.1.3. typedef void (* qapi_TIMER_cb_t)(uint32_t data)

Timer callback type. Timer callbacks should adhere to this signature.

15.1.2. Stucture Type

15.1.2.1. struct qapi_TIMER_define_attr_t

Timer define attribute type.

Type used to specify parameters when defining a timer.

```
typedef struct
{
   qbool_t     deferrable;
   qapi_TIMER_notify_t       cb_type;
   void*          sigs_func_ptr;
   uint32_t     sigs_mask_data;
} qapi_TIMER_define_attr_t;
```

- **Parameters**

Type	Parameter	Description
qbool_t	*deferrable*	FALSE=deferrable.
qapi_TIMER_notify_t	*cb_type*	Type of notification to receive.
void*	*sigs_func_ptr*	Specify the signal object or callback function.
uint32_t	*sigs_mask_data*	Specify the signal mask or callback data.

15.1.2.2. struct qapi_TIMER_set_attr_t

Timer set attribute type.

Type used to specify parameters when starting a timer.

```
typedef struct
{
  uint64_t     time;
  uint64_t     reload;
  qapi_TIMER_unit_type    unit;
} qapi_TIMER_set_attr_t;
```

- **Parameters**

Type	Parameter	Description
uint64_t	*time*	Timer duration.
uint64_t	*reload*	Reload duration.
qapi_TIMER_unit_type	*unit*	Specify units for timer duration.

15.1.2.3. struct qapi_TIMER_get_info_attr_t

Timer information attribute type.

Type used to get information for a given timer.

```
typedef struct
{
  void*        func_ptr;
  uint32_t    data;
} qapi_TIMER_get_cbinfo_t;
```

- **Parameters**

Type	Parameter	Description
void*	*func_ptr*	Specify the callback function.
uint32_t	*data*	Specify the callback data.

15.2. API Functions

15.2.1. qapi_Timer_Def

Allocates internal memory in the timer module. The internal memory is then formatted with parameters provided in the timer_attr variable. The timer_handle is returned to the client, and this handle must be used for any subsequent timer operations.

- **Prototype**

```
qapi_Status_t qapi_Timer_Def(qapi_TIMER_handle_t* timer_handle, qapi_TIMER_define_attr_t* timer_attr );
```

- **Parameters**

timer_handle:
[In] Handle to the timer

timer_attr:
[In] Attributes for defining the timer.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

15.2.2. qapi_Timer_Set

Starts the timer with the duration specified in timer_attr. If the timer is specified as a reload timer in timer_attr, the timer will restart after expiry.

- **Prototype**

```
qapi_Status_t qapi_Timer_Set ( qapi_TIMER_handle_t timer_handle, qapi_TIMER_set_attr_t * timer_attr );
```

- **Parameters**

timer_handle:
[In] Handle to the timer.

timer_attr:
[In] Attributes for setting the timer.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

The *qapi_Timer_Def()* API should be called for the timer before calling *qapi_Timer_Set()* function.

15.2.3. qapi_Timer_Get_Timer_Info

Gets the specified information about the timer.

- **Prototype**

```
qapi_Status_t qapi_Timer_Get_Timer_Info(qapi_TIMER_handle_t timer_handle, qapi_TIMER_get_info_attr_t * timer_info , uint64_t * data );
```

- **Parameters**

timer_handle:
[In] Handle to the timer.

timer_info:
[Out] Type of information needed from the timer.

data:
[Out] Returned timer information.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

15.2.4. qapi_Timer_Sleep

Blocks a thread for a specified time.

- **Prototype**

```
qapi_Status_t qapi_Timer_Sleep(uint64_t timeout, qapi_TIMER_unit_type unit, qbool_t non_deferrable );
```

- **Parameters**

timeout:
[In] Specify the duration to block the thread.

uint:
[In] Type of information needed from the timer.

non_deferrable:
[In] *TRUE* Indicates processor (if in deep sleep or power collapse) will be awakened on timeout.
FALSE Indicates processor will not be awakened from deep sleep or power collapse on timeout. Whenever the processor wakes up due to some other reason after timeout, the thread will be unblocked

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

15.2.5. qapi_Timer_Undef

Un-defines the timer. This API must be called whenever timer usage is done. Calling this API will release the internal memory of the timer that was allocated when the timer was defined.

- **Prototype**

```
qapi_Status_t qapi_Timer_Undef ( qapi_TIMER_handle_t timer_handle );
```

- **Parameters**

timer_handle:
[In] Timer handle for which to undefine the timer.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

15.2.6. qapi_Timer_Stop

Stops the timer.

- **Prototype**

```
qapi_Status_t qapi_Timer_Stop ( qapi_TIMER_handle_t timer_handle );
```

- **Parameters**

timer_handle:
[In] Timer handle for which to stop the timer.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16 Storage APIs

This chapter describes the file system data service and provides the following QAPIs:

```
qapi_FS_Open_With_Mode
qapi_FS_Open
qapi_FS_Read
qapi_FS_Write
qapi_FS_Close
qapi_FS_Rename
qapi_FS_Truncate
qapi_FS_Seek
qapi_FS_Mk_Dir
qapi_FS_Rm_Dir
qapi_FS_Unlink
qapi_FS_Stat
qapi_FS_Stat_With_Handle
qapi_FS_Statvfs
qapi_FS_Last_Error
```

16.1. Data Types

16.1.1. Enumeration Type

16.1.1.1. Enumration for Flag Bits to Open A File

Flag bits to open a file.

This enumeration is used in parameter *Oflag* of *qapi _FS_Open_With_Mode()* and *qapi_FS_Open()*.

```
enum
{
    QAPI_FS_O_RDONLY_E      = 1,
    QAPI_FS_O_WRONLY_E      = 2,
    QAPI_FS_O_RDWR_E        = 4,
    QAPI_FS_O_CREAT_E       = 8,
```

```
    QAPI_FS_O_EXCL_E        = 16,
    QAPI_FS_O_TRUNC_E       = 32,
    QAPI_FS_O_APPEND_E      = 64
};
```

- **Parameters**

Parameter	Description
QAPI_FS_O_RDONLY_E	Open for read only.
QAPI_FS_O_WRONLY_E	Open for write only.
QAPI_FS_O_RDWR_E	Open for read and write.
QAPI_FS_O_CREAT_E	Create the file if it does not exist.
QAPI_FS_O_EXCL_E	Fail if the file exists.
QAPI_FS_O_TRUNC_E	Truncate the file to zero bytes on successful open.
QAPI_FS_O_APPEND_E	All writes will self-seek to the end of the file before writing.

16.1.1.2. Enumration for Mode Bits to Open a File

Mode bits to open a file.

This enumeration is used in parameter *Mode* of *qapi_FS_Open_With_Mode().*

```
enum
{
    QAPI_FS_S_IRUSR_E    = 1,
    QAPI_FS_S_IWUSR_E    = 2,
    QAPI_FS_S_IXUSR_E    = 4,
    QAPI_FS_S_IRGRP_E    = 8,
    QAPI_FS_S_IWGRP_E    = 16,
    QAPI_FS_S_IXGRP_E    = 32,
    QAPI_FS_S_IROTH_E    = 64,
    QAPI_FS_S_IWOTH_E    = 128,
    QAPI_FS_S_IXOTH_E    = 256,
    QAPI_FS_S_ISUID_E    = 512,
    QAPI_FS_S_ISGID_E    = 1024,
    QAPI_FS_S_ISVTX_E    = 2048
};
```

- **Parameters**

Parameter	Description
QAPI_FS_S_IRUSR_E	Read permission for a user.
QAPI_FS_S_IWUSR_E	Write permission for a user.
QAPI_FS_S_IXUSR_E	Execute permission for a user.
QAPI_FS_S_IRGRP_E	Read permission for a group.
QAPI_FS_S_IWGRP_E	Write permission for a group.
QAPI_FS_S_IXGRP_E	Execute permission for a group.
QAPI_FS_S_IROTH_E	Read permission for others.
QAPI_FS_S_IWOTH_E	Write permission for others.
QAPI_FS_S_IXOTH_E	Execute permission for others.
QAPI_FS_S_ISUID_E	Set UID on execution.
QAPI_FS_S_ISGID_E	Set GID on execution.
QAPI_FS_S_ISVTX_E	Sticky bit (hidden attribute in FAT).

16.1.1.3. Enumration for Offset Bits to Seek a File

Offset bits to seek a file.

This enumeration is used in parameter *Whence* of *qapi_FS_Seek()*.

```
enum
{
    QAPI_FS_SEEK_SET_E      = 1,
    QAPI_FS_SEEK_CUR_E      = 2,
    QAPI_FS_SEEK_END_E      = 4
};
```

- **Parameters**

Parameter	Description
QAPI_FS_SEEK_SET_E	Set to offset.

QAPI_FS_SEEK_CUR_E	Set to offset+current position.
QAPI_FS_SEEK_END_E	Set to offset+file size.

16.1.1.4. enum qapi_FS_Filename_Rule_e

File name rules.

This enumeration is used in *qapi_FS_Statvfs_Type_s* structure.

```
enum qapi_FS_Filename_Rule_e
{
QAPI_FS_FILENAME_RULE_8BIT_RELAXED   = 1,
QAPI_FS_FILENAME_RULE_FAT            = 2,
QAPI_FS_FILENAME_RULE_FDI            = 3
};
```

- **Parameters**

Parameter	Description
QAPI_FS_FILENAME_RULE_8BIT_RELAXED	8-bit relaxed rule.
QAPI_FS_FILENAME_RULE_FAT	FAT rule.
QAPI_FS_FILENAME_RULE_FDI	FDI rule.

16.1.1.5. enum qapi_FS_Filename_Encoding_e

File name encoding schemes.

This enumeration is used in *qapi_FS_Statvfs_Type_s* structure.

```
enum qapi_FS_Filename_Encoding_e
{
    QAPI_FS_FILENAME_ENCODING_8BIT    = 1,
    QAPI_FS_FILENAME_ENCODING_UTF8    = 2
};
```

- **Parameters**

Parameter	Description
QAPI_FS_FILENAME_ENCODING_8BIT	8-bit encoding.
QAPI_FS_FILENAME_ENCODING_UTF8	UTF8 encoding.

16.1.2. Definition Type

16.1.2.1. QAPI Filesystem Macros

```
#define QAPI_FS_NAME_MAX          768
#define QAPI_FS_MAX_DESCRIPTORS   128
```

- **Parameters**

Parameter	Description
FS_NAME_MAX	Maximum length of a file name.
QAPI_FS_MAX_DESCRIPTORS	Maximum number of files that can be kept opened simultaneously.

16.1.3. Structure Type

16.1.3.1. struct qapi_FS_Stat_Type_s

Statistics type, used in the *qapi_FS_Stat()* API.

```
struct qapi_FS_Stat_Type_s
{
   uint16          st_dev;
   uint32          st_ino;
   uint16          st_Mode;
   uint8           st_nlink;
   uint32          st_size;
   unsigned long   st_blksize;
   unsigned long   st_blocks;
   uint32          st_atime;
   uint32          st_mtime;
   uint32          st_ctime;
```

```
    uint32          st_rdev;
    uint16          st_uid;
    uint16          st_gid;
};
```

- **Parameters**

Type	Parameter	Description
uint16	*st_dev*	Unique device ID among the mounted file systems.
uint32	*st_ino*	INode number associated with the file.
uint16	*st_Mode*	Mode associated with the file.
uint8	*st_nlink*	Number of active links that are referencing this file. The space occupied by the file is released after its references are reduced to 0.
uint32	*st_size*	File size in bytes.
unsigned long	*st_blksize*	Block size; smallest allocation unit of the file system. The unit in which the block count is represented.
unsigned long	*st_blocks*	Number of blocks allocated for this file in *st_blksize* units.
uint32	*st_atime*	Last access time. This is not updated, so it might have an incorrect value.
uint32	*st_mtime*	Last modification time. Currently, this indicates the time when the file was created.
uint32	*st_ctime*	Last status change time. Currently, this indicates the time when the file was created.
uint32	*st_rdev*	Major and minor device number for special device files.
uint16	*st_uid*	Owner or user ID of the file.
uint16	*st_gid*	Group ID of the file. The stored file data blocks are charged to the quota of this group ID.

16.1.3.2. Struct qapi_FS_Statvfs_Type_s

File system information, used in the *qapi_FS_Statvfs()* API.

```
struct qapi_FS_Statvfs_Type_s
{
    unsigned long                   f_bsize;
```

```
    uint32                              f_blocks;
    uint32                              f_bfree;
    uint32                              f_bavail;
    uint32                              f_files;
    uint32                              f_ffree;
    uint32                              f_favail;
    unsigned long                       f_fsid;
    unsigned long                       f_flag;
    unsigned long                       f_namemax;
    unsigned long                       f_maxwrite;
    uint32                              f_balloc;
    uint32                              f_hardalloc;
    unsigned long                       f_pathmax;
    unsigned long                       f_is_case_sensitive;
    unsigned long                       f_is_case_preserving;
    unsigned long                       f_max_file_size;
    unsigned long                       f_max_file_size_unit;
    unsigned long                       f_max_open_files;
    enum qapi_FS_Filename_Rule_e        f_name_rule;
    enum qapi_FS_Filename_Encoding_e    f_name_encoding;
};
```

- **Parameters**

Type	Parameter	Description
unsigned long	*f_bsize*	Fundamental file system block size. Minimum allocations in the file system happen at this size.
uint32	*f_blocks*	Maximum possible number of blocks available in the entire file system.
uint32	*f_bfree*	Total number of free blocks.
uint32	*f_bavail*	Number of free blocks available.
Uint32	*f_files*	Total number of file serial numbers.
uint32	*f_ffree*	Total number of free file serial numbers.
uint32	*f_favail*	Number of file serial numbers available.
unsigned long	*f_fsid*	File system ID; this varies depending on the implementation of the file system.
unsigned long	*f_flag*	Bitmask of f_flag values.

unsigned long	*f_namemax*	Maximum length of the name part of the string for a file, directory, or symlink.
unsigned long	*f_maxwrite*	Maximum number of bytes that can be written in a single write call.
uint32	*f_balloc*	Blocks allocated in the general pool
uint32	*f_hardalloc*	Hard allocation count. Resource intensive, so this is not usually computed.
unsigned long	*f_pathmax*	Maximum path length, excluding the trailing NULL. The unit here is in terms of character symbols. The number of bytes needed to represent a character will vary depending on the file name encoding scheme. For example, in a UTF8 encoding scheme, representing a single character could take anywhere between 1 to 4 bytes.
unsigned long	*f_is_case_sensitive*	Set to 1 if *Path* is case sensitive.
unsigned long	*f_is_case_preserving*	Set to 1 if *Path* is case preserved.
unsigned long	*f_max_file_size*	Maximum file size in the units determined by the member f_max_file-_size_unit.
unsigned long	*f_max_file_size_unit*	Unit type for f_max_file_size.
unsigned long	*f_max_open_files*	This member tells how many files can be kept opened for one particular file system. However, there is a global limit on how many files can be kept opened simultaneously across all file systems, which is configured by *QAPI_FS_MAX-_DESCRIPTORS.*
Enum qapi_FS_Filename_Rule_e	*f_name_rule*	File naming rule.
enum qapi_FS_Filename_Encoding_e	*f_name_encoding*	Encoding scheme.

16.2. API Functions

16.2.1. qapi_FS_Open_With_Mode

Opens a file as per the specified Oflag and mode.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Open_With_Mode ( const char * Path, int Oflag,qapi_FS_Mode_t Mode, int * Fd_ptr );
```

- **Parameters**

Path:
[In] Path of the file that is to be opened.

Oflag:
[In] Argument that describes how this file is to be opened. It contains one of the following values:

QAPI_FS_O_RDONLY_E	Open for read only.
QAPI_FS_O_WRONLY_E	Open for write only.
QAPI_FS_O_RDWR_E	Open for read and write.

In addition, the following flags can be bitwise ORed with this value:

QAPI_FS_O_APPEND_E	All writes will self-seek to the end of the file before writing.
QAPI_FS_O_CREAT_E	Create the file if it does not exist.
QAPI_FS_O_TRUNC_E	Truncate the file to zero bytes on successful open.

The following flags can be used to specify common ways of opening files:

QAPI_FS_O_CREAT_E | *QAPI_FS_O_TRUNC_E*	Normal for writing a file. Creates it if it does not exist. The resulting file is zero bytes long.
QAPI_FS_O_CREAT_E | *QAPI_FS_O_EXCL_E*	Creates a file but fails if it already exists.

Mode:
[In] If *QAPI_FS_O_CREAT* is a part of Oflag, a third argument (*Mode*) must be passed to *qapi_FS_open()* to define the file permissions given to the newly created file. If QAPI_FS_O_CREAT is not a part of flag, set *Mode*=0. One or more of the following permission bits can be ORed as the Mode parameter:

QAPI_FS_S_IRUSR_E	Read permission for a user
QAPI_FS_S_IWUSR_E	Write permission for a user
QAPI_FS_S_IXUSR_E	Execute permission for a user
QAPI_FS_S_IRGRP_E	Read permission for a group
QAPI_FS_S_IWGRP_E	Write permission for a group
QAPI_FS_S_IXGRP_E	Execute permission for a group
QAPI_FS_S_IROTH_E	Read permission for other
QAPI_FS_S_IWOTH_E	Write permission for other
QAPI_FS_S_IXOTH_E	Execute permission for other

QAPI_FS_S_ISUID_E	Set UID on execution
QAPI_FS_S_ISGID_E	Set GID on execution
QAPI_FS_S_ISVTX_E	Sticky bit (hidden attribute in FAT)

Fd_ptr:
[Out] Address of the file descriptor variable. On success, the file descriptor of an opened file is written to it. On failure, the variable is set to -1.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16.2.2. qapi_FS_Open

Opens a file as per the specified Oflag.

The parameters, error codes, and return types are the same as in the API *qapi_FS_Open_With_Mode()*. This function does not require the mode as an input argument. It opens the file in Default mode, which gives read and write permissions to the user, but not execute permissions.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Open ( const char * Path, int Oflag, int * Fd_ptr );
```

- **Parameters**

Path:
[In] Path of the file that is to be opened.

Oflag:
[In] Argument that describes how this file should be opened. See *qapi_FS_Open_With_Mode()*.

Fd_ptr:
[Out] Address of the file descriptor variable. On success, the file descriptor of an opened file is written to it. On failure, the variable is set to -1.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16.2.3. qapi_FS_Read

Attempts to read Count bytes of data from the file associated with the specified file descriptor.

Zero is a valid result and generally indicates that the end of the file has been reached. It is permitted for *qapi_FS_Read()* to return fewer bytes than were requested, even if the data is available in the file.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Read ( int Fd, uint8 * Buf, uint32 Count, uint32 *Bytes_Read_Ptr );
```

- **Parameters**

Fd:
[In] File descriptor obtained via the *qapi_FS_Open()* function.

Buf:
[Out] Buffer where the read bytes from the file will be stored.

Count:
[In] Number of bytes to read from the file.

Bytes_Read_Ptr:
[Out] This is a return from the function with the number of bytes actually read.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

A valid file descriptor should be obtained via *qapi_FS_Open()*.

16.2.4. qapi_FS_Write

Attempts to write *Count* bytes of data to the file associated with the specified file descriptor.

The write ioperation may happen at the current file pointer or at the end of the file if the file is opened with *QAPI_FS_O_APPEND*. It is permitted for *qapi_FS_Write* to write fewer bytes than were requested, even if space is available. If the number of bytes written is zero, it indicates that the file system is full (writing),

which will result in an *QAPI_ERR_ENOSPC* error.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Write ( int Fd, const uint8 * Buf, uint32 Count,uint32 * Bytes_Written_Ptr );
```

- **Parameters**

Fd:
[In] File descriptor obtained via the *qapi_FS_Open()* function.

Buf:
[In] Buffer to which the file is to be written.

Count:
[In] Number of bytes to write to the file.

Bytes_Read_Ptr:
[Out] This is a return from the function with the number of bytes actually written.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

A valid file descriptor should be obtained via *qapi_FS_Open()*.

16.2.5. qapi_FS_Close

Closes the file descriptor.

The descriptor will no longer refer to any file and will be allocated to subsequent calls to *qapi_FS_Open()*.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Close ( int Fd )
```

- **Parameters**

Fd:
[In] File descriptor obtained via the *qapi_FS_Open()* function.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

A valid file descriptor should be obtained via *qapi_FS_Open()*.

16.2.6. qapi_FS_Rename

Renames a file or a directory.

Files and directories (under the same file system) can be renamed. The arguments Old_Path and New_Path do not have to be in the same directory (but must be on the same file system device).

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Rename(const char* Old_Path, const char* New_Path);
```

- **Parameters**

Old_Path:
[In] Path name before the rename operation.

New_Path:
[In] Path name after the rename operation.

NOTE

qapi_FS_Rename() is atomic and will either successfully rename the file or leave the file in its original location.

- **Return Value**

QAPI_OK This function is executed successfully.
QAPI_ERROR It fails to execute this function.

- **Dependencies**

None.

16.2.7. qapi_FS_Truncate

Truncates a file to a specified length.

NOTE

If the supplied length is greater than the current file size, it depends on the underlying file system to determine whether the file can grow in size.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Truncate ( const char * Path, qapi_FS_Offset_t Length );
```

- **Parameters**

Path:
[In]Path of the file whose length is to be truncated.

Length:
[In] New size of the file. The bytes of length is in the range (-4 * 1024 *1024 * 1024, + 4 * 1024 * 1024 1024 -1).

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16.2.8. qapi_FS_Seek

Changes the file offset for the opened file descriptor.

Changing the file offset does not modify the file. If users lseek past the end of the file and then write, the gap will be filled with zero bytes. This gap may not actually allocate space. Using this API file can be seeked up to (4 GB -1) offset.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Seek ( int Fd, qapi_FS_Offset_t Offset, int Whence, qapi_FS_Offset_t * Actual_Offset_Ptr );
```

- **Parameters**

Fd:
[In] File descriptor obtained via the *qapi_FS_Open()* function.

Offset:
[In] New offset of the file.

Whence:
[In] Indicates how the new offset is computed:
QAPI_FS_SEEK_SET_E – Set to Offset.
QAPI_FS_SEEK_CUR_E – Set to Offset+current position.
QAPI_FS_SEEK_END_E – Set to Offset+file size.

Actual_Offset_Ptr:
[Out] Upon success, the resulting offset as bytes from the beginning of the file is filled in this parameter. On failure, the variable is set to -1.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

A valid file descriptor should be obtained via *qapi_FS_Open()*.

16.2.9. qapi_FS_Mk_Dir

Creates a new directory.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Mk_Dir(const char* path, qapi_FS_Mode_t mode);
```

- **Parameters**

Path:
[In] Path for the directory.

Mode:
[In] Permission bits of the new directory. See the *qapi_FS_Open()* API for information on Mode bits.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16.2.10. qapi_FS_Rm_Dir

Removes a directory. Only empty directories can be removed.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Rm_Dir ( const char * Path );
```

- **Parameters**

Path:
[In] Path of the directory that is to be removed.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16.2.11. qapi_FS_Unlink

Removes a link to a closed file.

If the link *Count* goes to zero, this will also remove the file. The *qapi_FS_Unlink()* API can be used on all file system objects except for directories. Use *qapi_FS_Rm_Dir()* for directories.

NOTE

The file must be closed for unlinking or removing. If it is open, the error *QAPI_ERR_ETXTBSY* is returned, indicating that the file is not closed.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Unlink ( const char * Path );
```

- **Parameters**

Path:
[In] File to which the link is to be removed.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16.2.12. qapi_FS_Stat

Returns the statistics of a file.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Stat ( const char * Path, struct qapi_FS_Stat_Type_s * SBuf );
```

- **Parameters**

Path:
[In] File descriptor of the file.

SBuf:
[In] For information on what is returned in the structure, see struct *qapi_FS_Stat_Type_s*.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

16.2.13. qapi_FS_Stat_With_Handle

Obtains information about the file with its open file handle.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Stat_With_Handle ( int Fd, struct qapi_FS_Stat_Type_s * SBuf );
```

- **Parameters**

Fd:
[In] Handle to a file obtained using the *qapi_FS_Open()* API.

SBuf:
[Out] Information is returned in the structure *qapi_FS_Stat_Type_s.*

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

A valid file descriptor should be obtained via *qapi_FS_Open()*.

16.2.14. qapi_FS_Statvfs

Obtains information about an entire file system.

Gets detailed information about the filesystem specified by the file path. Root or any mounted path for which to get information can be specified. If the root path is specified, information about the root file system is returned. Otherwise, information about the mounted file system specified by the path or the file system in which the given path exists is returned. The content details are in struct *qapi_FS_Statvfs_Type_s.*

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Statvfs ( const char * Path, struct qapi_FS_Statvfs_Type_s * SBuf );
```

- **Parameters**

Path:
[In] Valid path of a file or directory on the mounted file system.

SBuf:
[Out] Information is returned in the structure *qapi_FS_Statvfs_Type_s.*

- **Return Value**

QAPI_OK	This function is executed successfully.

QAPI_ERROR It fails to execute this function.

- **Dependencies**

None.

16.2.15. qapi_FS_Last_Error

Returns the last error that occurred in current task's context.

If it fails to execute q*api_FS_Open()* , an immediate call to *qapi_FS_Last_Error* returns the error for the failure. Otherwise, if another API, e.g., *qapi_FS_Read()*, is called after *qapi_FS_Open* failed within the same task, the error will be overwritten with *QAPI_OK* or a QAPI error code, depending whether *qapi_FS_Read()* was success or failed.

- **Prototype**

```
qapi_FS_Status_t qapi_FS_Last_Error ( void );
```

- **Parameters**

None.

- **Return Value**

QAPI_OK	No error in the last operation.
QAPI_ERR_EPERM	Operation is not permitted.
QAPI_ERR_EBADF	Bad file descriptor.
QAPI_ERR_EACCES	Permission denied.
QAPI_ERR_EXDEV	Attempt to cross the device.
QAPI_ERR_ENODEV	No such device.
QAPI_ERR_ENOTDIR	Not a directory.
QAPI_ERR_EISDIR	Is a directory.
QAPI_ERR_EMFILE	Too many open files.
QAPI_ERR_ETXTBSY	File or directory is already open.
QAPI_ERR_ENOSPC	No space is left on the device.
QAPI_ERR_ESPIPE	Seek is not permitted.
QAPI_ERR_ENAMETOOLONG	File name is too long.
QAPI_ERR_ENOTEMPTY	Directory is not empty.
QAPI_ERR_ELOOP	Too many symbolic links were encountered.
QAPI_ERR_EILSEQ	Illegal byte sequence.
QAPI_ERR_ESTALE	Stale remote file handle.
QAPI_ERR_EDQUOT	Attempt to write beyond the quota.
QAPI_ERR_EEOF	End of file.
QAPI_ERR_INVLD_ID	Invalid ID was passed by the kernel framework.
QAPI_ERR_UNKNOWN	Unknown error

- **Dependencies**

None.

17 Location APIs

This section provides data types and functions for the GNSS location driver.

```
qapi_Loc_Init
qapi_Loc_Deinit
qapi_Loc_Start_Tracking
qapi_Loc_Stop_Tracking
qapi_Loc_Update_Tracking_Options
qapi_Loc_Add_Geofences
qapi_Loc_Remove_Geofences
qapi_Loc_Modify_Geofences
qapi_Loc_Pause_Geofences
qapi_Loc_Resume_Geofences
qapi_Loc_Set_User_Buffer
```

17.1. Data Types

17.1.1. Enumeration Type

17.1.1.1. enum qapi_Location_Error_t

GNSS location error codes.

```
typedef enum
{
    QAPI_LOCATION_ERROR_SUCCESS = 0,
    QAPI_LOCATION_ERROR_GENERAL_FAILURE,
    QAPI_LOCATION_ERROR_CALLBACK_MISSING,
    QAPI_LOCATION_ERROR_INVALID_PARAMETER,
    QAPI_LOCATION_ERROR_ID_EXISTS,
    QAPI_LOCATION_ERROR_ID_UNKNOWN,
    QAPI_LOCATION_ERROR_ALREADY_STARTED,
    QAPI_LOCATION_ERROR_NOT_INITIALIZED,
    QAPI_LOCATION_ERROR_GEOFENCES_AT_MAX,
    QAPI_LOCATION_ERROR_NOT_SUPPORTED,
```

```
    QAPI_LOCATION_ERROR_TIMEOUT,
    QAPI_LOCATION_ERROR_LOAD_FAILURE,
    QAPI_LOCATION_ERROR_LOCATION_DISABLED,
    QAPI_LOCATION_ERROR_BEST_AVAIL_POS_INVALID,
} qapi_Location_Error_t;
```

- **Parameters**

Parameter	Description
QAPI_LOCATION_ERROR_SUCCESS	Success.
QAPI_LOCATION_ERROR_GENERAL_FAILURE	General failure.
QAPI_LOCATION_ERROR_CALLBACK_MISSING	Callback is missing.
QAPI_LOCATION_ERROR_INVALID_PARAMETER	Invalid parameter.
QAPI_LOCATION_ERROR_ID_EXISTS	ID already exists.
QAPI_LOCATION_ERROR_ID_UNKNOWN	ID is unknown.
QAPI_LOCATION_ERROR_ALREADY_STARTED	Already started.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	Not initialized.
QAPI_LOCATION_ERROR_GEOFENCES_AT_MAX	Maximum number of Geofences reached.
QAPI_LOCATION_ERROR_NOT_SUPPORTED	Not supported.
QAPI_LOCATION_ERROR_TIMEOUT	Timeout when asking single shot.
QAPI_LOCATION_ERROR_LOAD_FAILURE	GNSS engine could not get loaded.
QAPI_LOCATION_ERROR_LOCATION_DISABLED	Location module license is disabled.
QAPI_LOCATION_ERROR_BEST_AVAIL_POS_INVALID	Best available position is invalid.

17.1.1.2. enum qapi_Location_Flags_t

Flags to indicate which values are valid in a location. This enumeration is used in *qapi_Location_t* structure.

```
t typedef enum
{
    QAPI_LOCATION_HAS_LAT_LONG_BIT    =    (1 << 0),
```

```
    QAPI_LOCATION_HAS_ALTITUDE_BIT      =    (1 << 1),
    QAPI_LOCATION_HAS_SPEED_BIT         =    (1 << 2),
    QAPI_LOCATION_HAS_BEARING_BIT       =    (1 << 3),
    QAPI_LOCATION_HAS_ACCURACY_BIT      =    (1 << 4),
    QAPI_LOCATION_HAS_VERTICAL_ACCURACY_BIT     =    (1 << 5),
    QAPI_LOCATION_HAS_SPEED_ACCURACY_BIT        =    (1 << 6),
    QAPI_LOCATION_HAS_BEARING_ACCURACY_BIT      =    (1 << 7),
    QAPI_LOCATION_HAS_ALTITUDE_MSL_BIT    =    (1 << 8),
    QAPI_LOCATION_IS_BEST_AVAIL_POS_BIT   =    (1 << 9),
} qapi_Location_Flags_t;
```

- **Parameters**

Parameter	Description
QAPI_LOCATION_HAS_LAT_LONG_BIT	Location has a valid latitude and longitude.
QAPI_LOCATION_HAS_ALTITUDE_BIT	Location has a valid altitude.
QAPI_LOCATION_HAS_SPEED_BIT	Location has a valid speed.
QAPI_LOCATION_HAS_BEARING_BIT	Location has a valid bearing.
QAPI_LOCATION_HAS_ACCURACY_BIT	Location has valid accuracy.
QAPI_LOCATION_HAS_VERTICAL_ACCURACY_BIT	Location has valid vertical accuracy.
QAPI_LOCATION_HAS_SPEED_ACCURACY_BIT	Location has valid speed accuracy.
QAPI_LOCATION_HAS_BEARING_ACCURACY_BIT	Location has valid bearing accuracy.
QAPI_LOCATION_HAS_ALTITUDE_MSL_BIT	Location has valid altitude wrt mean sea level.
QAPI_LOCATION_IS_BEST_AVAIL_POS_BIT	Location is the currently best available position.

17.1.1.3. enum qapi_Geofence_Breach_Mask_Bits_t

Flags to indicate Geofence breach mask bit.

This enumeration is used in *qapi_Geofence_Option_t* and *qapi_Geofence_Info_t* structure.

```
typedef enum
{
    QAPI_GEOFENCE_BREACH_ENTER_BIT =        (1 << 0),
```

```
    QAPI_GEOFENCE_BREACH_EXIT_BIT =          (1 << 1),
    QAPI_GEOFENCE_BREACH_DWELL_IN_BIT =    (1 << 2),
    QAPI_GEOFENCE_BREACH_DWELL_OUT_BIT = (1 << 3),
} qapi_Geofence_Breach_Mask_Bits_t;
```

- **Parameters**

Parameter	Description
QAPI_GEOFENCE_BREACH_ENTER_BIT	Breach enter bit.
QAPI_GEOFENCE_BREACH_EXIT_BIT	Breach exit bit.
QAPI_GEOFENCE_BREACH_DWELL_IN_BIT	Breach dwell in bit.
QAPI_GEOFENCE_BREACH_DWELL_OUT_BIT	Breach dwell out bit.

17.1.1.4. enum qapi_Location_Capabilities_Mask_Bits_t

Flags to indicate the capabilities bit. This enumeration is used in typedef *void(*qapi_Capabilities_Callback)(qapi_Location_Capabilities_Mask_t capabilitiesMask).*

```
typedef enum
{
    QAPI_LOCATION_CAPABILITIES_TIME_BASED_TRACKING_BIT =        (1 << 0),
    QAPI_LOCATION_CAPABILITIES_TIME_BASED_BATCHING_BIT =        (1 << 1),
    QAPI_LOCATION_CAPABILITIES_DISTANCE_BASED_TRACKING_BIT = (1 << 2),
    QAPI_LOCATION_CAPABILITIES_DISTANCE_BASED_BATCHING_BIT = (1 << 3),
    QAPI_LOCATION_CAPABILITIES_GEOFENCE_BIT =                  (1 << 4),
    QAPI_LOCATION_CAPABILITIES_GNSS_DATA_BIT =                 (1 << 5),
} qapi_Location_Capabilities_Mask_Bits_t;
```

- **Parameters**

Parameter	Description
QAPI_LOCATION_CAPABILITIES_TIME_BASED_TRACKING_BIT	Capabilities time-based tracking bit.
QAPI_LOCATION_CAPABILITIES_TIME_BASED_BATCHING_BIT	Capabilities time-based batching bit.
QAPI_LOCATION_CAPABILITIES_DISTANCE_BASED_TRACKING_BIT	Capabilities distance-based tracking bit.
QAPI_LOCATION_CAPABILITIES_DISTANCE_BASED_BATCHING_BIT	Capabilities distance-based batching

	bit.
QAPI_LOCATION_CAPABILITIES_GEOFENCE_BIT	Capabilities Geofence bit.
QAPI_LOCATION_CAPABILITIES_GNSS_DATA_BIT	Capabilities GNSS data bit.

17.1.1.5. enum qapi_Location_Accuracy_Level_t

Flags to indicate the desired level of accuracy for fix computation. This enumeration is used in *qapi_Location_Options_t* structure.

```
typedef enum
{
QAPI_LOCATION_ACCURACY_UNKNOWN = 0,
QAPI_LOCATION_ACCURACY_LOW,
QAPI_LOCATION_ACCURACY_MED,
QAPI_LOCATION_ACCURACY_HIGH,
} qapi_Location_Accuracy_Level_t;
```

- **Parameters**

Parameter	Description
QAPI_LOCATION_ACCURACY_UNKNOWN	Accuracy is not specified, use default.
QAPI_LOCATION_ACCURACY_LOW	Low Accuracy for location is acceptable.
QAPI_LOCATION_ACCURACY_MED	Medium Accuracy for location is acceptable.
QAPI_LOCATION_ACCURACY_HIGH	Only High Accuracy for location is acceptable.

17.1.2. Typdefs

17.1.2.1. typedef void(*qapi_Capabilities_Callback)(qapi_Location_Capabilities_Mask_t capabilitiesMask)

Provides the capabilities of the system. It is called once after *qapi_Loc_Init()* is called.

- **Prototype**

```
typedef void(*qapi_Capabilities_Callback)(qapi_Location_Capabilities_Mask_t capabilitiesMask);
```

- **Parameters**

capabilitiesMask:
[In] Bitwise OR of *qapi_Location_Capabilities_Mask_Bits_t*

- **Return Value**

None.

17.1.2.2. typedef void(*qapi_Response_Callback)(qapi_Location_Error_t err,uint32_t id)

Response callback, which is used by all tracking, batching, and Geofence APIs. It is called for every tracking, batching, and Geofence API.

- **Prototype**

```
typedef void(*qapi_Response_Callback)(qapi_Location_Error_t err,uint32_t id);
```

- **Parameters**

err:
[In] *qapi_Location_Error_t* associated with the request.
If this is not *QAPI_LOCATION_ERROR_SUCCESS*, the ID is not valid.

id:
[In] ID to be associated with the request.

- **Return Value**

None.

17.1.2.3. typedef void(*qapi_Collective_Response_Callback)(size_t count,

qapi_Location_Error_t* err, uint32_t* ids)

Collective response callback is used by Geofence APIs. It is called for every Geofence API call.

- **Prototype**

```
typedef void(*qapi_Collective_Response_Callback)( size_t count, qapi_Location_Error_t* err, uint32_t* ids);
```

- **Paramerters**

count:
[In] Number of locations in arrays.

err:
[In] Array of *qapi_Location_Error_t* associated with the request.

ids:
[In] Array of IDs to be associated with the request.

- **Return Value**

None.

17.1.2.4. typedef void (*qapi_Tracking_Callback)(qapi_Location_t location)

Tracking callback used for the *qapi_Loc_Start_Tracking()* API. This is an optional function and can be NULL. It is called when delivering a location in a tracking session.

- **Prototype**

```
typedef void(*qapi_Tracking_Callback)( qapi_Location_t location);
```

- **Parameters**

location:
[In] Structure containing information related to the tracked location.

- **Return Value**

None.

17.1.2.5. typedef void (*qapi_Batching_Callback)(size_t count, qapi_Location_t* location)

This is an optional function and can be NULL. It is called when delivering a location in a batching session.

- **Prototype**

```
typedef void(*qapi_Batching_Callback)( size_t count, qapi_Location_t* location);
```

- **Parameters**

count:
[In] Number of locations in an array.

location:
[In] Array of location structures containing information related to the batched locations.

- **Return Value**

None.

17.1.2.6. typedef void(*qapi_Capabilities_Callback)(qapi_Location_Capabilities_Mask_t capabilitiesMask)

Geofence breach callback used for the *qapi_Loc_Add_Geofences()* API. This is an optional function and can be NULL. It is called when any number of geofences have a state change.

- **Prototype**

```
typedef void(*qapi_Geofence_Breach_Callback)( qapi_Geofence_Breach_Notification_t geofenceBreachNotification);
```

- **Parameters**

geofenceBreachNotification:
[In] Breach notification information.

- **Return Value**

None.

17.1.2.7. typedef void(*qapi_Single_Shot_Callback)(qapi_Location_t location, qapi_Location_Error_t err)

This is an optional function and can be NULL. It is called when delivering a location in a single shot session broadcasted to all clients, no matter if a session has started by client.

- **Prototype**

```
typedef void(*qapi_Single_Shot_Callback)( qapi_Location_t location, qapi_Location_Error_t err);
```

- **Parameters**

location:
[In] Structure containing information related to the tracked location.

err:
[In] *qapi_Location_Error_t* associated with the request. This could be *QAPI_LOCATION_ERROR_-SUCCESS* (location is valid) or *QAPI_LOCATION_ERROR_TIMEOUT* (a timeout occurred, location is not valid).

- **Return Value**

None.

17.1.2.8. typedef void (*qapi_Gnss_Data_Callback)(qapi_Gnss_Data_t gnssData)

This is an optional function and can be NULL. It is called when delivering a GNSS Data structure. The GNSS data structure contains useful information (e.g., jammer indication). This callback will be called every second.

- **Prototype**

```
typedef void(*qapi_Gnss_Data_Callback)( qapi_Gnss_Data_t gnssData);
```

- **Parameters**

gnssData:
[In] Structure containing information related to the requested GNSS Data

- **Return Value**

None.

17.1.2.9. typedef void(*qapi_Location_Meta_Data_Callback)(qapi_Location_Meta_Data_t metaData)

This is an optional function and can be NULL. It is called when delivering some location meta data.

- **Prototype**

```
typedef void(*qapi_Location_Meta_Data_Callback)( qapi_Location_Meta_Data_t metaData);
```

- **Parameters**

metaData:
[In] Structure containing meta data related to ongoing location sessions.

- **Return Value**

None.

17.1.2.10. typedef void (*qapi_Gnss_Nmea_Callback)(qapi_Gnss_Nmea_t gnssNmea)

NMEA callback used for reporting NMEA statements. This is an optional function and can be NULL. It is called when delivering NMEA report from Modem.

- **Prototype**

```
typedef void(*qapi_Gnss_Nmea_Callback)( qapi_Gnss_Nmea_t gnssNmea);
```

- **Parameters**

gnssNmea:
[In] Structure containing the NMEA sentence.

- **Return Value**

None.

17.1.2.11. typedef void(*qapi_Motion_Tracking_Callback)(qapi_Location_Motion_Info_t motionInfo)

This is an optional function and can be NULL. It is called when delivering motion info in a motion tracking session.

- **Prototype**

```
typedef void(*qapi_Motion_Tracking_Callback)( qapi_Location_Motion_Info_t motionInfo);
```

- **Parameters**

motionInfo:
[In] Structure containing information about the detected motion.

- **Return Value**

None.

17.1.3. Stucture Type

17.1.3.1. struct qapi_Location_t

Structure for location information.

```
typedef struct
{
    size_t size;
    qapi_Location_Flags_Mask_t flags;
    uint64_t timestamp;
    double latitude;
    double longitude;
    double altitude;
    double altitudeMeanSeaLevel;
    float speed;
    float bearing;
    float accuracy;
    float verticalAccuracy;
    float speedAccuracy;
    float bearingAccuracy;
} qapi_Location_t;
```

- **Parameters**

Type	Parameter	Description
size_t	*size*	Size. Set to the size of *qapi_Location_t*.
qapi_Location_Flags_Mask_t	*flags*	Bitwise OR of *qapi_Location_Flags_t*.
uint64_t	*timestamp*	UTC timestamp for a location fix; milliseconds since Jan. 1, 1970.
double	*latitude*	Latitude in degrees.

double	*longitude*	Longitude in degrees.
double	*altitude*	Altitude in meters above the WGS 84 reference ellipsoid.
double	*altitudeMeanSeaLevel*	Altitude in meters with respect to mean sea level.
float	*speed*	Speed in meters per second.
float	*bearing*	Bearing in degrees, range: 0-360.
float	*accuracy*	Accuracy in meters.
float	*verticalAccuracy*	Vertical accuracy in meters.
float	*speedAccuracy*	Speed accuracy in meters/second.
float	*bearingAccuracy*	Bearing accuracy in degrees (0 to 359.999).

17.1.3.2. struct qapi_Location_Options_t

Structure for location options.

```
typedef struct
{
    size_t size;
    uint32_t minInterval;
    uint32_t minDistance;
    qapi_Location_Accuracy_Level_t    accuracyLevel;
} qapi_Location_Options_t;
```

- **Parameters**

Type	Parameter	Description
size_t	*size*	Size. Set to the size of *qapi_Location_Options_t*.
uint32_t	*minInterval*	There are three different interpretations of this field, depending if minDistance is 0 or not: 1. Time-based tracking (minDistance = 0). minInterval is the minimum time interval in milliseconds that must elapse between final position reports. 2. Distance-based tracking (minDistance > 0). minInterval is the maximum time period in

		milliseconds after the minimum distance criteria has been met within which a location update must be provided. If set to 0, an ideal value will be assumed by the engine. 3. Batching. minInterval is the minimum time interval in milliseconds that must elapse between position reports.
uint32_t	*minInterval*	Minimum distance in meters that must be traversed between position reports. Setting this interval to 0 will be a pure time-based tracking/batching.
qapi_Location_Accuracy_Level_t	*accuracyLevel*	Accuracy level required for fix computation.

17.1.3.3. struct qapi_Geofence_Option_t

Structure for Geofence options.

```
typedef struct
{
    size_t size;
    qapi_Geofence_Breach_Mask_t breachTypeMask;
    uint32_t responsiveness;
    uint32_t dwellTime;
} qapi_Geofence_Option_t;
```

- **Parameters**

Type	Parameter	Description
size_t	*size*	Size. Set to the size of *qapi_Geofence_Option_t.*
qapi_Geofence_Breach_Mask_t	*breachTypeMask*	Bitwise OR of *qapi_Geofence_Breach_Mask_Bits_t* bits.
uint32_t	*responsiveness*	Specifies in milliseconds the user-defined rate of detection for a Geofencebreach. This may impact the time lag between the actual breach event andwhen it is reported. The gap between the actual breach and the time it is reported depends on the user setting. The power implication is inversely proportional to the responsiveness value set by the user.The higher the responsiveness value, the lower the power implications, and vice-versa.

uint32_t	*dwellTime*	Dwell time is the time in milliseconds a user spends in the Geofence before a dwell event is sent.

17.1.3.4. struct qapi_Geofence_Info_t

Structure for Geofence information.

```
typedef struct
{
size_t size;
double latitude;
double longitude;
double radius;
} qapi_Geofence_Info_t;
```

- **Parameters**

Type	Parameter	Description
size_t	*size*	Size. Set to the size of *qapi_Geofence_Info_t.*
double	*latitude*	Bitwise OR of *qapi_Geofence_Breach_Mask_Bits_t* bits.
double	*longitude*	Longitude of the center of the Geofence in degrees.
double	*radius*	Radius of the Geofence in meters.

17.1.3.5. struct qapi_Geofence_Breach_Notification_t

Structure for Geofence breach notification.

```
typedef struct
{
    size_t size;
    size_t count;
    uint32_t* ids;
    qapi_Location_t location;
    qapi_Geofence_Breach_t type;
    uint64_t timestamp;
} qapi_Geofence_Breach_Notification_t;
```

- **Parameters**

Type	Parameter	Description
size_t	*size*	Size. Set to the size of *qapi_Geofence_Breach_Notification_t.*
size_t	*count*	Number of IDs in the array.
uint32_t *	*ids*	Array of IDs that have been breached.
qapi_Location_t	*location*	Location associated with a breach.
qapi_Geofence_Breach_t	*type*	Type of breach.
uint64_t	*timestamp*	Timestamp of the breach; milliseconds since Jan. 1, 1970.

17.1.3.6. struct qapi_Location_Callbacks_t

Location callbacks requirements.

```
typedef struct
{
    size_t size;
    qapi_Capabilities_Callback          capabilitiesCb;
    qapi_Response_Callback                responseCb;
    qapi_Collective_Response_Callback   collectiveResponseCb;
    qapi_Tracking_Callback                trackingCb;
    qapi_Batching_Callback                batchingCb;
    qapi_Geofence_Breach_Callback         geofenceBreachCb;
    qapi_Single_Shot_Callback             singleShotCb;
    qapi_Gnss_Data_Callback               gnssDataCb;
    qapi_Location_Meta_Data_Callback    metaDataCb;
    qapi_Gnss_Nmea_Callback               gnssNmeaCb;
    qapi_Motion_Tracking_Callback       motionTrackingCb;
} qapi_Location_Callbacks_t;
```

- **Parameters**

Type	Parameter	Description
size_t	*size*	Size. Set to the size of *qapi_Location_Callbacks_t*.
qapi_Capabilities_Callback	*capabilitiesCb*	Capabilities callback is mandatory.

qapi_Response_Callback	*responseCb*	Response callback is mandatory.
qapi_Collective_Response_Callback	*collectiveResponseCb*	Collective response callback is mandatory.
qapi_Tracking_Callback	*trackingCb*	Tracking callback is optional.
qapi_Batching_Callback	*batchingCb*	Batching callback is optional.
qapi_Geofence_Breach_Callback	*geofenceBreachCb*	Geofence breach callback is optional.
qapi_Single_Shot_Callback	*singleShotCb*	Single shot callback is optional.
qapi_Gnss_Data_Callback	*gnssDataCb*	GNSS data callback is optional.
qapi_Location_Meta_Data_Callback	*metaDataCb*	Meta data callback is optional.
qapi_Gnss_Nmea_Callback	*gnssNmeaCb*	NMEA callback used for reporting NMEA statements.
qapi_Motion_Tracking_Callback	*motionTrackingCb*	Motion tracking callback is optional.

17.2. API Functions

17.2.1. qapi_Loc_Init

Initializes a location session and registers the callbacks.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Init ( qapi_loc_client_id * pClientId, const qapi_Location_Callbacks_t * pCallbacks );
```

- **Description**

Initializes a location session and registers the callbacks.

- **Parameters**

pCallbacks:
[In] Pointer to the structure with the callback functions to be registered.

pClientId:
[Out] Pointer to client ID type, where the unique identifier for this location client is returned.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_GENERAL_FAILURE	There is an internal error.
QAPI_LOCATION_ERROR_CALLBACK_MISSING	One of the mandatory callback functions is missing.
QAPI_LOCATION_ERROR_ALREADY_STARTED	A location session has already been initialized.

- **Dependencies**

None.

17.2.2. qapi_Loc_Deinit

De-initializes a location session.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Deinit ( qapi_loc_client_id clientId );
```

- **Parameters**

clientId:
[In] Pointer to client ID type, where the unique identifier for this location client is returned.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	No location session has been initialized.

- **Dependencies**

A valid ID of location client must be obtained by *qapi_Loc_Init().*

17.2.3. qapi_Loc_Start_Tracking

Starts a tracking session, which returns a session ID that will be used by the other tracking APIs and in the response callback to match the command with a response. Locations are reported on the tracking callback passed in *qapi_Loc_Init()* periodically according to the location options.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Start_Tracking ( qapi_loc_client_id clientId, const
qapi_Location_Options_t * pOptions, uint32_t * pSessionId );
```

- **Parameters**

clientId :
[In] Pointer to client ID type, where the unique identifier for this location client is returned.

pOptions:
[In] Pointer to a structure containing the options:

minInterval
- There are two different interpretations of this field, depending on the value of minDistance: ime-based tracking (minDistance=0). minInterval is the minimum time interval in milliseconds that must elapse between final position reports. Distance-based tracking (minDistance>0). minInterval is the maximum time period in milliseconds after the minimum distance criteria has been met within which a location update must be provided. If set to 0, an ideal value is assumed by the engine.

minDistance
– Minimum distance in meters that must be traversed between position reports. Setting this interval to 0 results in purely time-based tracking.

pSessionId:
[Out] Pointer to the session ID to be returned.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	(0x00) The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	(0x07) No location session has been initialized.

- **Dependencies**

A valid ID of location client must be obtained by *qapi_Loc_Init().*

17.2.4. qapi_Loc_Stop_Tracking

Stops a tracking session associated with the id parameter.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Stop_Tracking ( qapi_loc_client_id clientId,uint32_t sessionId );
```

- **Parameters**

clientId:
[In] Pointer to client ID type, where the unique identifier for this location client is returned.

sessionId:
[In] ID of the session to be stopped.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	(0x00) The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	(0x07) No location session has been initialized.

- **Dependencies**

A valid session ID must be obtained by *qapi_Loc_Start_Tracking().*

17.2.5. qapi_Loc_Update_Tracking_Options

Changes the location options of a tracking session associated with the ID parameter.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Update_Tracking_Options ( qapi_loc_client_id clientId, uint32_t sessionId, const qapi_Location_Options_t *pOptions );
```

- **Parameters**

clientId :
[In] Pointer to client ID type, where the unique identifier for this location client is returned.

sessionId:
[In] ID of the session to be changed.

pOptions:
[In] Pointer to a structure containing the options:

minInterval
– There are two different interpretations of thisfield, depending on the value of minDistance:
Time-based tracking (minDistance=0). minInterval is the minimum time interval in milliseconds that must elapse between final position reports.Distance-based tracking (minDistance>0). minInterval is the maximum time period in milliseconds after the minimum distance criteria has been met within which a location update must be provided. If set to 0, an ideal value is assumed by the engine.

minDistance
– Minimum distance in meters that must be traversed between position reports. Setting this interval to 0 results in purely time-based tracking.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	(0x00) The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	(0x07) No location session has been initialized.

- **Dependencies**

A valid session ID must be obtained by *qapi_Loc_Start_Tracking().*

17.2.6. qapi_Loc_Add_Geofences

Adds a specified number of Geofences and returns an array of Geofence IDs that will be used by the other Geofence APIs, as well as in the Geofence response callback to match the command with a response. The Geofence breach callback delivers the status of each Geofence according to the Geofence options for each.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Add_Geofences ( qapi_loc_client_id clientId, size_t count, const qapi_Geofence_Option_t * pOptions, const qapi_Geofence_Info_t * pInfo, uint32_t ** pIdArray );
```

- **Parameters**

clientId:
[In] Client identifier for the location client.

count:
[In] Number of Geofences to be added.

pOptions:
[In] Array of structures containing the options:
breachTypeMask – Bitwise OR of *qapi_Geofence_Breach_Mask_Bits_t* bits
responsiveness in milliseconds
dwellTime in seconds.

pInfo:
[In] Array of structures containing the data:
Latitude of the center of the Geofence in degrees
Longitude of the center of the Geofence in degrees
Radius of the Geofence in meters.

pIdArray:
[Out] Array of IDs of Geofences to be returned.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	No location session has been initialized.

- **Dependencies**

A valid ID of location client must be obtained by *qapi_Loc_Init().*

17.2.7. qapi_Loc_Remove_Geofences

Removes a specified number of Geofences.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Remove_Geofences ( qapi_loc_client_id clientId, size_t count, const uint32_t * pIDs );
```

- **Parameters**

clientId:
[In] Client identifier for the location client.

count:
[In] Number of Geofences to be removed.

pIDs:
[In] Array of IDs of the Geofences to be removed.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	No location session has been initialized.

- **Dependencies**

A valid Number of Geofences must be add by *qapi_Loc_Add_Geofences ()*

17.2.8. qapi_Loc_Modify_Geofences

Gets a number of locations that are currently stored or batched on the low power processor, delivered by the batching callback passed to *qapi_Loc_Init()*. Locations are then deleted from the batch stored on the low power processor.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Modify_Geofences(qapi_loc_client_id clientId, size_t count,
const uint32_t* pIDs, const qapi_Geofence_Option_t* options);
```

- **Parameters**

clientId:
[In] Client identifier for the location client.

count:
[In] Number of Geofences to be modified

pIDs:
[In] Array of IDs of the Geofences to be modified.

pOptions:
[In] Array of structures containing the options:
breachTypeMask – Bitwise OR of GeofenceBreachTypeMask bits
responsiveness in milliseconds
dwellTime in seconds.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	No location session has been initialized.

- **Dependencies**

A valid Number of Geofences must be add by *qapi_Loc_Add_Geofences()*.

17.2.9. qapi_Loc_Pause_Geofences

Pauses a specified number of Geofences, which is similar to *qapi_Loc_Remove_Geofences()* except that they can be resumed at any time.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Pause_Geofences ( qapi_loc_client_id clientId, size_t count, const uint32_t * pIDs);
```

- **Parameters**

clientId:
[In] Client identifier for the location client.

count:
[In] Number of Geofences to be paused.

pIDs:
[In] Array of IDs of the Geofences to be paused.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	No location session has been initialized.

- **Dependencies**

A valid Number of Geofences must be add by *qapi_Loc_Add_Geofences ().*

17.2.10. qapi_Loc_Resume_Geofences

Resumes a specified number of Geofences that are paused.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Resume_Geofences ( qapi_loc_client_id clientId, size_t count, const uint32_t * pIDs );
```

- **Parameters**

clientId:
[In] Client identifier for the location client.

count:
[In] Number of Geofences to be resumed.

pIDs:
[In] Array of IDs of the Geofences to be resumed

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED	No location session has been initialized.

- **Dependencies**

A valid Number of Geofences must be add by *qapi_Loc_Add_Geofences()*.

17.2.11. qapi_Loc_Set_User_Buffer

Sets the user buffer to be used for sending back callback data.

- **Prototype**

```
qapi_Location_Error_t qapi_Loc_Set_User_Buffer(qapi_loc_client_id clientId,uint8_t* pUserBuffer, size_t bufferSize);
```

- **Parameters**

clientId:
[In] Client ID for which user buffer is to be set.

pUserBuffer:
[In] User memory buffer to hold information passedin callbacks. Note that since buffer is shared by all the callbacks this has to be consumed at the user side before it can be used by another callback to avoid any potential race condition.

bufferSize:
[In] Size of user memory buffer to hold information passed in callbacks. This size should be large enough to accomodate the largest data size passed in a callback.

- **Return Value**

QAPI_LOCATION_ERROR_SUCCESS	The operation was successful.
QAPI_LOCATION_ERROR_GENERAL_FAILURE	There is an internal error.

- **Dependencies**

A valid ID of location client must be obtained by *qapi_Loc_Init().*

18 AT Forward Service APIs

AT forward service framework is used for customers to customize AT commands in their application, and the maximum number of custom commands is 15.

This chapter provides the following APIs:

```
qapi_atfwd_reg
qapi_atfwd_dereg
qapi_atfwd_send_resp
qapi_atfwd_send_urc_resp
```

18.1. API Functions

18.1.1. qapi_atfwd_reg

Registers new custom AT commands along with respective callbacks.

- **Prototype**

```
qapi_Status_t qapi_atfwd_reg ( char * name, at_fwd_cb_type atfwd_callback );
```

- **Parameters**

name:
[In] Pointer to an AT commands string.

atfwd_callback:
[In] Pointer to the callback corresponding to the AT commands.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

None.

18.1.2. qapi_atfwd_dereg

Deregisters AT commands.

- **Prototype**

```
qapi_Status_t qapi_atfwd_dereg(char *name);
```

- **Parameters**

name:
[In] Pointer to an AT commands string.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

The AT command to be deregistered should be registered via *qapi_atfwd_reg()* firstly.

18.1.3. qapi_atfwd_send_resp

Sends a response.

- **Prototype**

```
qapi_Status_t qapi_atfwd_send_resp(char *atcmd_name, char *buf, uint32_t result);
```

- **Parameters**

atcmd_name:
[In] Pointer to the particular AT command to which this response corresponds.

buf:
[In] Pointer to the buffer containing the response.

result:
[In] *0* Result ERROR. This is to be set in case of ERROR or CME ERROR. The response buffer contains the entire details.
1 Result OK. This is to be set if the final response is to send an OK result code to the terminal.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

The particular AT command must be registered via *qapi_atfwd_reg()* firstly.

18.1.4. qapi_atfwd_send_urc_resp

Sends a URC response.

- **Prototype**

```
qapi_Status_t qapi_atfwd_send_urc_resp(char *atcmd_name, char *at_urc);
```

- **Parameters**

atcmd_name:
[In] Pointer to the particular AT command to which this response corresponds.

at_urc:
[In] Pointer to the buffer containing the response.

- **Return Value**

QAPI_OK	This function is executed successfully.
QAPI_ERROR	It fails to execute this function.

- **Dependencies**

The particular AT command must be registered via *qapi_atfwd_reg()* firstly.

19 QAPI Status and Error Codes

This chapter describes definitions of common and some module-specific status and error codes.

19.1. QAPI Modules and Error Codes Definition Format

19.1.1. QAPI Modules

The following definitions represent the IDs for the various modules of the QAPI, these QAPI Module IDs are used to define QAPI status and error codes.

```
#define QAPI_MOD_BASE                   (0)
#define QAPI_MOD_802_15_4               (1)
#define QAPI_MOD_NETWORKING             (2)
#define QAPI_MOD_WIFI                   (3)
#define QAPI_MOD_BT                     (4)
#define QAPI_MOD_BSP                    (5)
#define QAPI_MOD_BSP_I2C_MASTER         (6)
#define QAPI_MOD_BSP_SPI_MASTER         (7)
#define QAPI_MOD_BSP_TLMM               (8)
#define QAPI_MOD_BSP_GPIOINT            (9)
#define QAPI_MOD_BSP_PWM                (10)
#define QAPI_MOD_BSP_ERR                (11)
#define QAPI_MOD_BSP_DIAG               (12)
#define QAPI_MOD_BSP_OM_SMEM            (13)
#define QAPI_MOD_CRYPTO                 (14)
#define QAPI_MOD_BSP_FS                 (15)
#define QAPI_MOD_BSP_USB                (16)
#define QAPI_MOD_BSP_FTL                (17)
#define QAPI_MOD_RIL                    (18)
#define QAPI_MOD_BSP_ADC                (19)
#define QAPI_MOD_BSP_TSENS              (20)
#define QAPI_MOD_BSP_PMIC               (21)
```

19.1.2. Error Codes Definition Format

The following definitions are used to format error codes based on their module. Error codes that use these macros will be a negative value of the format -((10000 * <Module ID>) + <Status Code>).

```
#define __QAPI_ERR_MOD_OFFSET (10000)
#define __QAPI_ERR_ENCAP_MOD_ID(__mod_id__) ((__mod_id__) * __QAPI_ERR_MOD_OFFSET)
#define __QAPI_ERROR(__mod_id__, __err__) (0 - (__QAPI_ERR_ENCAP_MOD_ID(__mod_id__) + (__err__)))
```

19.2. Common QAPI Status Codes

The following definitions represent the status codes common to all of the QAPI modules.

```
#define  QAPI_OK                    ((qapi_Status_t)(0))
#define  QAPI_ERROR                 ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 1)))
#define  QAPI_ERR_INVALID_PARAM     ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 2)))
#define  QAPI_ERR_NO_MEMORY         ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 3)))
#define  QAPI_ERR_NO_RESOURCE       ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 4)))
#define  QAPI_ERR_BUSY              ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 6)))
#define  QAPI_ERR_NO_ENTRY          ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 7)))
#define  QAPI_ERR_NOT_SUPPORTED     ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 8)))
#define  QAPI_ERR_TIMEOUT           ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 9)))
#define  QAPI_ERR_BOUNDS            ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE,10)))
#define  QAPI_ERR_BAD_PAYLOAD       ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE,11)))
#define  QAPI_ERR_EXISTS            ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE,12)))
#define  QAPI_ERR_NOT_INITIALIZED   ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE,13)))
#define  QAPI_ERR_INVALID_STATE     ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE,13)))
#define  QAPI_ERR_API_DEPRACATED    ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE,14)))
```

- **Parameters**

Parameter	Description
QAPI_OK	Success.
QAPI_ERROR	General error.
QAPI_ERR_INVALID_PARAM	Invalid parameter.
QAPI_ERR_NO_MEMORY	Memory allocation error.
QAPI_ERR_NO_RESOURCE	Resource allocation error.

QAPI_ERR_BUSY	Operation busy.
QAPI_ERR_NO_ENTRY	Entry not found.
QAPI_ERR_NOT_SUPPORTED	Feature not supported.
QAPI_ERR_TIMEOUT	Operation timed out.
QAPI_ERR_BOUNDS	Out of bounds.
QAPI_ERR_BAD_PAYLOAD	Bad Payload.
QAPI_ERR_EXISTS	Entry already exists.
QAPI_ERR_NOT_INITIALIZED	Uninitialized.
QAPI_ERR_INVALID_STATE	Invalid state.
QAPI_ERR_API_DEPRACATED	QAPI function is deprecated.

19.3. Generic Error Codes

The following definitions represent the generic error codes.

```
#define  QAPI_NET_ERR_INVALID_IPADDR
((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,  21)))

#define  QAPI_NET_ERR_CANNOT_GET_SCOPEID
((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,  22)))

#define  QAPI_NET_ERR_SOCKET_CMD_TIME_OUT
((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,  23)))

#define  QAPI_NET_ERR_MAX_SERVER_REACHED
((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,  24)))
```

- **Parameters**

Parameter	Description
QAPI_NET_ERR_INVALID_IPADDR	IP address is invalid.
QAPI_NET_ERR_CANNOT_GET_SCOPEID	Failed to get the scope ID.

QAPI_NET_ERR_SOCKET_CMD_TIME_OUT	Socket command timed out.
QAPI_NET_ERR_MAX_SERVER_REACHED	Maximum server address (v4/v6) reached in the server's list.

19.4. MQTT Error Codes

The following definitions represent the MQTT error codes.

```
#define  QAPI_NET_MQTT_ERR_NUM_START 25

#define  QAPI_NET_MQTT_ERR_ALLOC_FAILURE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START)

#define  QAPI_NET_MQTT_ERR_BAD_PARAM
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 1))

#define  QAPI_NET_MQTT_ERR_BAD_STATE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 2))

#define  QAPI_NET_MQTT_ERR_CONN_CLOSED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 3))

#define  QAPI_NET_MQTT_ERR_MSG_DESERIALIZATION_FAILURE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 4))

#define  QAPI_NET_MQTT_ERR_MSG_SERIALIZATION_FAILURE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 5))

#define  QAPI_NET_MQTT_ERR_NEGATIVE_CONNACK
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 6))

#define  QAPI_NET_MQTT_ERR_NO_DATA
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 7))
```

```
#define  QAPI_NET_MQTT_ERR_NONZERO_REFCOUNT
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 8))

#define  QAPI_NET_MQTT_ERR_PINGREQ_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 9))

#define  QAPI_NET_MQTT_ERR_PUBACK_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 10))

#define  QAPI_NET_MQTT_ERR_PUBCOMP_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 11))

#define  QAPI_NET_MQTT_ERR_PUBLISH_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 12))

#define  QAPI_NET_MQTT_ERR_PUBREC_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 13))

#define  QAPI_NET_MQTT_ERR_PUBREL_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 14))

#define  QAPI_NET_MQTT_ERR_RX_INCOMPLETE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 15))

#define  QAPI_NET_MQTT_ERR_SOCKET_FATAL_ERROR
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 16))

#define  QAPI_NET_MQTT_ERR_TCP_BIND_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 17))

#define  QAPI_NET_MQTT_ERR_TCP_CONNECT_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 18))
```

```
#define  QAPI_NET_MQTT_ERR_SSL_CONN_FAILURE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 19))

#define  QAPI_NET_MQTT_ERR_SUBSCRIBE_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 21))

#define  QAPI_NET_MQTT_ERR_SUBSCRIBE_UNKNOWN_TOPIC
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 21))

#define  QAPI_NET_MQTT_ERR_UNSUBSCRIBE_MSG_CREATION_FAILED
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 22))

#define  QAPI_NET_MQTT_ERR_UNEXPECTED_MSG
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 23))

#define  QAPI_NET_MQTT_ERR_PARTIAL_SUBSCRIPTION_FAILURE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 24))

#define  QAPI_NET_MQTT_ERR_RESTORE_FAILURE
((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING,  QAPI_NET_MQTT_ERR_NUM_START
+ 25))

#define  QAPI_NET_MQTT_ERR_MAX_NUMS  26
```

- **Parameters**

Parameter	Description
QAPI_NET_MQTT_ERR_NUM_START	MQTT error number start.
QAPI_NET_MQTT_ERR_ALLOC_FAILURE	MQTT memory allocation failed.
QAPI_NET_MQTT_ERR_BAD_PARAM	MQTT bad parameter while invoking the API.
QAPI_NET_MQTT_ERR_BAD_STATE	MQTT connection is in a bad state.
QAPI_NET_MQTT_ERR_CONN_CLOSED	MQTT connection is closed.

QAPI_NET_MQTT_ERR_MSG_DESERIALIZATION_FAILURE	MQTT packet decode failed.
QAPI_NET_MQTT_ERR_MSG_SERIALIZATION_FAILURE	MQTT packet encode failed.
QAPI_NET_MQTT_ERR_NEGATIVE_CONNACK	MQTT negative CONNACK recevied.
QAPI_NET_MQTT_ERR_NO_DATA	MQTT no data.
QAPI_NET_MQTT_ERR_NONZERO_REFCOUNT	MQTT no zero reference count while disconnecting.
QAPI_NET_MQTT_ERR_PINGREQ_MSG_CREATION_FAILED	MQTT ping request message creation failed.
QAPI_NET_MQTT_ERR_PUBACK_MSG_CREATION_FAILED	MQTT PUBACK message creation failed.
QAPI_NET_MQTT_ERR_PUBCOMP_MSG_CREATION_FAILED	MQTT PUBCOM message creation failed.
QAPI_NET_MQTT_ERR_PUBLISH_MSG_CREATION_FAILED	MQTT publish message creation failed.
QAPI_NET_MQTT_ERR_PUBREC_MSG_CREATION_FAILED	MQTT PUBREC message creation failed.
QAPI_NET_MQTT_ERR_PUBREL_MSG_CREATION_FAILED	MQTT PUBREL message creation failed.
QAPI_NET_MQTT_ERR_RX_INCOMPLETE	MQTT Rx is incomplete.
QAPI_NET_MQTT_ERR_SOCKET_FATAL_ERROR	MQTT socket fatal error.
QAPI_NET_MQTT_ERR_TCP_BIND_FAILED	MQTT TCP bind error.
QAPI_NET_MQTT_ERR_TCP_CONNECT_FAILED	MQTT TCP connection error.
QAPI_NET_MQTT_ERR_SSL_CONN_FAILURE	MQTT SSL connection failed.
QAPI_NET_MQTT_ERR_SUBSCRIBE_MSG_CREATION_FAILED	MQTT subscribe message creation failed.
QAPI_NET_MQTT_ERR_SUBSCRIBE_UNKNOWN_TOPIC	MQTT subscribe unknown topic.
QAPI_NET_MQTT_ERR_UNSUBSCRIBE_MSG_CREATION_FAILED	MQTT unsubscribe message creation failed.
QAPI_NET_MQTT_ERR_UNEXPECTED_MSG	MQTT unexpected message received.
QAPI_NET_MQTT_ERR_PARTIAL_SUBSCRIPTION_FAILURE	MQTT subscription failed.
QAPI_NET_MQTT_ERR_RESTORE_FAILURE	MQTT restore failed.
QAPI_NET_MQTT_ERR_MAX_NUMS	MQTT maximum error number.

19.5. SSL Error Codes

The following definitions represent the SSL error codes.

```
#define QAPI_ERR_SSL_CERT          __QAPI_ERROR(QAPI_MOD_NETWORKING, 1)
#define QAPI_ERR_SSL_CONN          __QAPI_ERROR(QAPI_MOD_NETWORKING, 2)
#define QAPI_ERR_SSL_HS_NOT_DONE   __QAPI_ERROR(QAPI_MOD_NETWORKING, 3)
#define QAPI_ERR_SSL_ALERT_RECV    __QAPI_ERROR(QAPI_MOD_NETWORKING, 4)
#define QAPI_ERR_SSL_ALERT_FATAL   __QAPI_ERROR(QAPI_MOD_NETWORKING, 5)
#define QAPI_SSL_HS_IN_PROGRESS    __QAPI_ERROR(QAPI_MOD_NETWORKING, 6)
#define QAPI_SSL_OK_HS             __QAPI_ERROR(QAPI_MOD_NETWORKING, 7)
#define QAPI_ERR_SSL_CERT_CN       __QAPI_ERROR(QAPI_MOD_NETWORKING, 8)
#define QAPI_ERR_SSL_CERT_TIME     __QAPI_ERROR(QAPI_MOD_NETWORKING, 9)
#define QAPI_ERR_SSL_CERT_NONE     __QAPI_ERROR(QAPI_MOD_NETWORKING, 10)
#define QAPI_ERR_SSL_NETBUF        __QAPI_ERROR(QAPI_MOD_NETWORKING, 11)
#define QAPI_ERR_SSL_SOCK          __QAPI_ERROR(QAPI_MOD_NETWORKING, 12)
```

- **Parameters**

Parameter	Description
QAPI_ERR_SSL_CERT	Error in own certificate.
QAPI_ERR_SSL_CONN	Error with the SSL connection.
QAPI_ERR_SSL_HS_NOT_DONE	Handshake must be completed before the operation can be attempted.
QAPI_ERR_SSL_ALERT_RECV	Received an SSL warning alert message.
QAPI_ERR_SSL_ALERT_FATAL	Received an SSL fatal alert message.
QAPI_SSL_HS_IN_PROGRESS	Handshake is in progress.
QAPI_SSL_OK_HS	Handshake was successful.
QAPI_ERR_SSL_CERT_CN	The SSL certificate of the peer is trusted, CN matches the host name, time has expired.
QAPI_ERR_SSL_CERT_TIME	The SSL certificate of the peer is trusted, CN does not match the host name, time is valid.
QAPI_ERR_SSL_CERT_NONE	The SSL certificate of the peer is not trusted.
QAPI_ERR_SSL_NETBUF	Connection drops when out of network buffers; usually a resource configuration error.
QAPI_ERR_SSL_SOCK	Socket error; use qapi_errno.h to check for the reason code.

19.6. HTTP(S) Error Codes

The following enumerations represent the HTTP(S) error codes.

```
typedef enum
{
  HTTPC_ERR_SSL_CFG = -10,
  HTTPC_ERR_SOCKET_OPEN = -9,
  HTTPC_ERR_SSL_CONN = -8,
  HTTPC_ERR_INVALID_PARAM = -7,
  HTTPC_ERR_BUSY = -6,
  HTTPC_ERR_NO_MEMORY = -5,
  HTTPC_ERR_SEND = -4,
  HTTPC_ERR_CONN = -3,
  HTTPC_ERR_NONE_SESS = -2,
  HTTPC_ERROR = -1,
  HTTPC_OK = 0,
} http_client_error_code_e;
```

20 Appendix A References

Table 1: Terms and Abbreviations

Abbreviation	Description
API	Application Programming Interface
AES	Advanced Encryption Standard
APN	Access Point Name
BSD	Berkeley Software Distribution
CA	Certificate Authority
CBC	Cipher Block Chaining
CE	Call End
CMD	Comand
CS	Chip selection
DHE	Diffie Hellman Ephemeral
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name or System
DSS	Data Services Sockets
DTLS	Datagram Transport Layer Security
ECDH	Elliptic Curve Diffie–Hellman key Exchange
ECDSA	Elliptic Curve Digital Signature Algorithm
EMM	Enterprise Mobile Management
EMMS	Enterprise Mobile Management state

FAT	File Allocation Table
FDI	Floppy Disk image
FDD	Frequency Division Duplexing
GPIO	General-purpose Input/Output
HTTP	Hyper Text Transfer Protocol
HW	Hardware
GCM	Galois/Counter Mode
GSM	Global System for Mobile Communications
ICCID	Integrate circuit card identity
ICMP	Internet Control Message Protocol
IMEI	International Mobile Equipment Identity
IMSI	International Mobile Subscriber Identity
I2C	Inter-Integrated Circuit
LTE	Long Term Evolution
MCC	Mobile Country Code
MDN	Mobile Directory Number
MTU	Maximum Transmission Unit
MQTT	Message Queuing Telemetry Transport
MNC	Mobile Network Code
NB_IoT	Narrow Band Internet of Things
OS	Operating System
OOB	Out of Band
PCI	Physical-layer Cell Identity
PDP	Packet Data Protocol
PDN	Public Data Network

PMM	Pin Mode Multiplexer
PSK	Pre-shared Key
RRC	Radio Resource Control
RSRQ	Reference Signal Receiving Quality
RTOS	Real Time Operating System
RX	Receive
TCP	Transmission Control Protocol
TDD	Time Division Duplex
TTL	Time to Live
TX	Transmit
QAPI	Qualcomm™ Application Programming Interface
QMI	Qualcomm Messaging Interface
QOS	Quality of Service
SACK	Selective ACK
SHA	Secure Hash Algorithm
SPI	Serial Peripheral Interface
SIM	Subscriber Identification Module
SSL	Secure Sockets Layer
SoC	System on Chip
TLS	Transport Layer Security
UART	Universal Asynchronous Receiver/Transmitter
UDP	User Datagram Protocol
URC	Unsolicited Result Code
URL	Uniform Resource Locator
WCDMA	Wideband Code Division Multiple Access

Table 2: TLS/DTLS Supported Ciphersuites

Ciphersuite	Defined ciphersuite's name	TLS1.2/ DTLS1.2 supported ciphers	TLS1.1, TLS1.0, or DTLS1.0 supported ciphers only
	TLS_NULL_WITH_NULL_NULL	No	No
PSK (preshared keys)	TLS_PSK_WITH_RC4_128_SHA	No	No
	TLS_PSK_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_PSK_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_PSK_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_PSK_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_PSK_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_PSK_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_PSK_WITH_AES_256_CBC_SHA384	Yes	No
ECDHE_ECDSA (Ephemeral Elliptic Curve Diffie-Hellman with Elliptic Curve Digital Signature Algorithm key)	TLS_ECDHE_ECDSA_WITH_NULL_SHA	No	No
	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	No	No
	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	Yes	No
ECDH_ECDSA (Elliptic Curve Diffie-Hellman with Elliptic Curve Digital Signature Algorithm key)	TLS_ECDH_ECDSA_WITH_NULL_SHA	No	No
	TLS_ECDH_ECDSA_WITH_RC4_128_SHA	No	No
	TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	Yes	No
ECDHE_RSA	TLS_ECDHE_RSA_WITH_NULL_SHA	No	No
	TLS_ECDHE_RSA_WITH_RC4_128_SHA	No	No
	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Yes	No

	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	Yes	No
ECDH_RSA	TLS_ECDH_RSA_WITH_NULL_SHA	No	No
	TLS_ECDH_RSA_WITH_RC4_128_SHA	No	No
	TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	Yes	No
DHE_RSA (Diffie-Hellman signed using RSA keys)	TLS_DHE_RSA_WITH_DES_CBC_SHA	No	No
	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_DHE_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	Yes	No
	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_DHE_RSA_WITH_AES_128_CCM	Yes	No
	TLS_DHE_RSA_WITH_AES_256_CCM	Yes	No
	TLS_DHE_RSA_WITH_AES_128_CCM_8	Yes	No
	TLS_DHE_RSA_WITH_AES_256_CCM_8	Yes	No
	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	Yes	No
RSA	TLS_RSA_WITH_NULL_MD5	No	No
	TLS_RSA_WITH_NULL_SHA	No	No
	TLS_RSA_WITH_RC4_128_MD5	No	No
	TLS_RSA_WITH_RC4_128_SHA	No	No
	TLS_RSA_WITH_DES_CBC_SHA	Yes	Yes
	TLS_RSA_WITH_3DES_EDE_CBC_SHA	No	Yes
	TLS_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_RSA_WITH_NULL_SHA256	No	No
	TLS_RSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_RSA_WITH_AES_256_CBC_SHA256	Yes	No
	TLS_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_RSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_RSA_WITH_AES_128_CCM	Yes	No
	TLS_RSA_WITH_AES_256_CCM	Yes	No
	TLS_RSA_WITH_AES_128_CCM_8	Yes	No

	TLS_RSA_WITH_AES_256_CCM_8	Yes	No