



PERFECT WIRELESS EXPERIENCE

FIBOCOM MA510 Series Basic QAPI Application Guide

Version: V1.0.0

Date: 2021-02-03



Applicability Type

No.	Product Model	Description
1	MA510-GL-00-90	MA510 open API

FIBOCOM
Confidential



Copyright

Copyright ©2021 Fibocom Wireless Inc. All rights reserved.

Without the prior written permission of the copyright holder, any company or individual is prohibited to excerpt, copy any part of or the entire document, or transmit the document in any form.

Notice

The document is subject to update from time to time owing to the product version upgrade or other reasons. Unless otherwise specified, the document only serves as the user guide. All the statements, information and suggestions contained in the document do not constitute any explicit or implicit guarantee.

Trademark



The trademark is registered and owned by Fibocom Wireless Inc.

Change History

Version	Author	Reviewer	Approver	Date	Description
V1.0.0	gaoyongshun	Zhangpei	sean	2021-01-04	Initial version

FIBOCOM
Confidential

Contents

1	DSS Net Control APIs	12
1.1	DSS Netctrl Data Types	13
1.1.1	Data Define	18
1.1.2	Data Structure	27
1.1.3	Data Typedef	40
1.1.4	Data Enumeration	41
1.2	API Functions	45
1.2.1	qapi_DSS_Init	45
1.2.2	qapi_DSS_Release	46
1.2.3	qapi_DSS_Get_Data_Srvc_Hndl	47
1.2.4	qapi_DSS_Rel_Data_Srvc_Hndl	48
1.2.5	qapi_DSS_Set_Data_Call_Param	49
1.2.6	qapi_DSS_Start_Data_Call	50
1.2.7	qapi_DSS_Stop_Data_Call	51
1.2.8	qapi_DSS_Get_Pkt_Stats	52
1.2.9	qapi_DSS_Reset_Pkt_Stats	52
1.2.10	qapi_DSS_Get_Call_End_Reason	54
1.2.11	qapi_DSS_Get_Call_Tech	55
1.2.12	qapi_DSS_Get_Current_Data_Bearer_Tech	56
1.2.13	qapi_DSS_Get_Device_Name	57
1.2.14	qapi_DSS_Get_Qmi_Port_Name	58
1.2.15	qapi_DSS_Get_IP_Addr_Count	59
1.2.16	qapi_DSS_Get_IP_Addr	60
1.2.17	qapi_DSS_Get_IP_Addr_Per_Family	61
1.2.18	qapi_DSS_Get_Link_Mtu	62
1.2.19	qapi_DSS_Get_Link_Mtu_per_family	63
1.2.20	qapi_DSS_Add_MO_Exception_IPdata_Filters	64
1.2.21	qapi_DSS_Remove_MO_Exception_IPdata_Filters	65
1.2.22	qapi_DSS_Nipd_Send	66
1.2.23	qapi_DSS_Set_Transfer_Status_v2	67
1.2.24	qapi_DSS_Get_Apn_Rate_Control	68
1.2.25	qapi_DSS_Get_Splmn_Rate_Control	69
2	QAPI Networking Socket	70
2.1	QAPI Data Types	72
2.1.1	Data Define	77
2.1.2	Data Structure	93
2.2	API Functions	98
2.2.1	qapi_socket	98
2.2.2	qapi_bind	99
2.2.3	qapi_listen	100
2.2.4	qapi_accept	101
2.2.5	qapi_connect	102

2.2.6	qapi_setsockopt	103
2.2.7	qapi_getsockopt	104
2.2.8	qapi_socketclose	105
2.2.9	qapi_errno	106
2.2.10	qapi_recvfrom	107
2.2.11	qapi_recv	108
2.2.12	qapi_sendto	109
2.2.13	qapi_send	110
2.2.14	qapi_select	111
2.2.15	qapi_select_v2	112
2.2.16	qapi_fd_zero	113
2.2.17	qapi_fd_zero_v2	114
2.2.18	qapi_fd_clr	115
2.2.19	qapi_fd_clr_v2	116
2.2.20	qapi_fd_set	117
2.2.21	qapi_fd_set_v2	118
2.2.22	qapi_fd_isset	119
2.2.23	qapi_fd_isset_v2	120
2.2.24	qapi_getpeername	121
2.2.25	qapi_getsockname	122
3	QAPI Network Security APIs	123
3.1.1	Data Define	124
3.1.2	Data Structure	135
3.1.3	Data Typedef	140
3.1.4	Data Enumeration	142
3.1	API Functions	144
3.1.1	qapi_Net_SSL_Obj_New	144
3.1.2	qapi_Net_SSL_Con_New	145
3.1.3	qapi_Net_SSL_Configure	146
3.1.4	qapi_Net_SSL_Cert_delete	148
3.1.5	qapi_Net_SSL_Cert_Store	149
3.1.6	qapi_Net_SSL_Cert_Convert_And_Store	150
3.1.7	qapi_Net_SSL_Cert_Load	151
3.1.8	qapi_Net_SSL_Cert_Load_Get_Identifier	152
3.1.9	qapi_Net_SSL_Cert_List	153
3.1.10	qapi_Net_SSL_Fd_Set	154
3.1.11	qapi_Net_SSL_Accept	155
3.1.12	qapi_Net_SSL_Connect	156
3.1.13	qapi_Net_SSL_Shutdown	157
3.1.14	qapi_Net_SSL_Obj_Free	158
3.1.15	qapi_Net_SSL_Read	159
3.1.16	qapi_Net_SSL_Write	160
3.1.17	qapi_Net_SSL_Cert_File_Exists	161
3.1.18	qapi_Net_SSL_Set_Extended_Config_Option	162

4	QAPI Networking Services	163
4.1	Networking Data Types.....	164
4.1.1	Data Define	164
4.1.2	Data Structure	170
4.1.3	Data Enumeration	177
4.2	API Functions.....	179
4.2.1	qapi_Net_Get_All_Ifnames	179
4.2.2	inet_pton	180
4.2.3	const char* inet_ntop.....	181
4.2.4	qapi_Net_Interface_Get_Physical_Address.....	182
4.2.5	qapi_Net_Interface_Exist.....	183
4.2.6	qapi_Net_IPv4_Config.....	184
4.2.7	qapi_Net_Ping	185
4.2.8	qapi_Net_Ping_2	186
4.2.9	qapi_Net_IPv4_Route.....	187
4.2.10	qapi_Net_Ping6	188
4.2.11	qapi_Net_Ping6_2	189
4.2.12	qapi_Net_IPv6_Get_Address	190
4.2.13	qapi_Net_IPv6_Route.....	191
4.2.14	qapi_Net_IPv6_Get_Scope_ID.....	192
5	Domain Name System Client Service APIs	193
5.1	DNS Data Types	194
5.1.1	Data Define	194
5.1.2	Data Structure	197
5.1.3	Data Enumeration	198
5.2	API Functions.....	199
5.2.1	qapi_Net_DNSc_Set_Client_timeout.....	199
5.2.2	qapi_Net_DNSc_Is_Started	200
5.2.3	qapi_Net_DNSc_Command	201
5.2.4	qapi_Net_DNSc_Reshost	202
5.2.5	qapi_Net_DNSc_Reshost_on_iface.....	203
5.2.6	qapi_Net_DNSc_Get_Server_List	204
5.2.7	qapi_Net_DNSc_Get_Server_Index	205
5.2.8	qapi_Net_DNSc_Add_Server.....	206
5.2.9	qapi_Net_DNSc_Add_Server_on_iface.....	207
5.2.10	qapi_Net_DNSc_Del_Server.....	208
5.2.11	qapi_Net_DNSc_Del_Server_on_iface	209
5.2.12	qapi_Net_DNSc_Host_By_Name	210
5.2.13	qapi_Net_DNSc_Host_By_Name2	211
6	MQTT APIs.....	212
6.1	MQTT Data Types.....	213
6.1.1	Data Define	214
6.1.2	Data Structure	216
6.1.3	Data Typedef.....	220

6.1.4	Data Enumeration	221
6.2	API Functions	224
6.2.1	qapi_Net_MQTT_New	224
6.2.2	qapi_Net_MQTT_Destroy	225
6.2.3	qapi_Net_MQTT_Connect	226
6.2.4	qapi_Net_MQTT_Disconnect	227
6.2.5	qapi_Net_MQTT_Publish	228
6.2.6	qapi_Net_MQTT_Publish_Get_Msg_Id	229
6.2.7	qapi_Net_MQTT_Subscribe	230
6.2.8	qapi_Net_MQTT_Unsubscribe	231
6.2.9	qapi_Net_MQTT_Set_Connect_Callback	232
6.2.10	qapi_Net_MQTT_Set_Subscribe_Callback	233
6.2.11	qapi_Net_MQTT_Set_Message_Callback	234
6.2.12	qapi_Net_MQTT_Set_Publish_Callback	235
6.2.13	qapi_Net_MQTT_Allow_Unsub_Publish	236
6.2.14	qapi_Net_MQTT_Set_Extended_Config_Option	237
6.2.15	qapi_Net_MQTT_Deserialize_Publish_Header	238
7	HTTP(S) APIs	240
7.1	HTTP(S) Data Types	241
7.1.1	Data Define	241
7.1.2	Data Structure	242
7.1.3	Data Typedef	244
7.1.4	Data Enumeration	245
7.1	API Functions	247
7.2.1	qapi_Net_HTTPc_Start	247
7.2.2	qapi_Net_HTTPc_Stop	248
7.2.3	qapi_Net_HTTPc_New_sess	249
7.2.4	qapi_Net_HTTPc_Free_sess	250
7.2.5	qapi_Net_HTTPc_Connect	251
7.2.6	qapi_Net_HTTPc_Proxy_Connect	252
7.2.7	qapi_Net_HTTPc_Disconnect	253
7.2.8	qapi_Net_HTTPc_Request	254
7.2.9	qapi_Net_HTTPc_Set_Body	255
7.2.10	qapi_Net_HTTPc_Set_Param	256
7.2.11	qapi_Net_HTTPc_Add_Header_Field	257
7.2.12	qapi_Net_HTTPc_Clear_Header	258
7.2.13	qapi_Net_HTTPc_Configure_SSL	259
7.2.14	qapi_Net_HTTPc_Configure	260
7.2.15	qapi_Net_HTTPc_Extended_Config_Options	261
8	Device Information Module	262
8.1	Device Information	263
8.1.1	Data Define	263
8.1.2	Data Structure	272
8.1.3	Data Typedef	324

8.1.4	Data Enumeration	325
8.2	API Functions	343
8.2.1	qapi_Device_Info_Init_v2	343
8.2.2	qapi_Device_Info_Get_v2	344
8.2.3	qapi_Device_Info_Set_Callback_v2	345
8.2.4	qapi_Device_Info_Release_v2	346
8.2.5	qapi_Device_Info_Reset_v2	347
8.2.6	qapi_Device_Info_Request	348
8.2.7	qapi_Device_Info_Clear_Callback_v2	350
9	GPIO Interrupt Controller APIs	351
9.1	GPIO Data Types	352
9.1.1	Data Typedef	352
9.1.2	Data Enumeration	353
9.2	API Functions	354
9.2.1	qapi_GPIoint_Register_Interrupt	354
9.2.2	qapi_GPIoint_Deregister_Interrupt	356
9.2.3	qapi_GPIoint_Set_Trigger	357
9.2.4	qapi_GPIoint_Enable_Interrupt	358
9.2.5	qapi_GPIoint_Disable_Interrupt	359
9.2.6	qapi_GPIoint_Trigger_Interrupt	360
9.2.7	qapi_GPIoint_Is_Interrupt_Pending	361
10	PMM APIs	362
10.1	PMM Data Types	363
10.1.1	Data Structure	363
10.1.2	Data Typedef	364
10.1.3	Data Enumeration	365
10.2	API Functions	367
10.2.1	qapi_TLMM_Get_Gpio_ID	367
10.2.2	qapi_TLMM_Release_Gpio_ID	368
10.2.3	qapi_TLMM_Config_Gpio	369
11	Pulse Width Modulation (PWM)	370
11.1	PWM Data Types	371
11.1.1	Data Typedef	371
11.1.2	Data Enumeration	371
11.2	API Functions	372
11.2.1	qapi_PWM_Get_ID	372
11.2.2	qapi_PWM_Release_ID	373
11.2.3	qapi_PWM_Enable	374
11.2.4	qapi_PWM_Set_Frequency	375
11.2.5	qapi_PWM_Set_Duty_Cycle	377
11.2.6	qapi_PWM_Get_Clock_Frequency	379
11.2.7	qapi_TLMM_Drive_Gpio	380
11.2.8	qapi_TLMM_Read_Gpio	381
12	I2C Master APIs	382

12.1	I2C Date Types	383
12.1.1	Data Define	383
12.1.2	Data Structure	384
12.1.3	Data Typedef	386
12.1.4	DataEnumeration	387
12.2	API Functions	389
12.2.1	qapi_I2CM_Open	389
12.2.2	qapi_I2CM_Close	390
12.2.3	qapi_I2CM_Transfer	391
12.2.4	qapi_I2CM_Power_On	393
12.2.5	qapi_I2CM_Power_Off	394
13	SPI Master APIs	395
13.1	SPI Date Types	396
13.1.1	Data Structure	396
13.1.2	Data Typedef	398
13.1.3	Data Enumeration	399
13.2	API Functions	402
14	UART APIs	408
14.1	SPI Date Types	409
14.2	API Functions	415
15	Storage Module	423
15.1	File System Data Types	424
15.1.1	Data Define	424
15.1.2	Data Structure	425
15.1.3	Data Typedef	428
15.1.4	Data Enumeration	429
15.2	API Functions	431
15.2.1	qapi_FS_Open_With_Mode	431
15.2.2	qapi_FS_Open	434
15.2.3	qapi_FS_Read	435
15.2.4	qapi_FS_Write	436
15.2.5	qapi_FS_Close	437
15.2.6	qapi_FS_Rename	438
15.2.7	qapi_FS_Truncate	439
15.2.8	qapi_FS_Seek	440
15.2.9	qapi_FS_Mk_Dir	441
15.2.10	qapi_FS_Rm_Dir	442
15.2.11	qapi_FS_Unlink	443
15.2.12	qapi_FS_Stat	444
15.2.13	qapi_FS_Stat_With_Handle	445
15.2.14	qapi_FS_Statvfs	446
15.2.15	qapi_FS_Iter_Open	447
15.2.16	qapi_FS_Iter_Next	448
15.2.17	qapi_FS_Iter_Close	450

15.2.18	qapi_FS_Last_Error	451
15.2.19	qapi_FTL_Open	452
15.2.20	qapi_FTL_Close	454
15.2.21	qapi_FTL_Get_info	455
15.2.22	qapi_FTL_Read_lpa	456
15.2.23	qapi_FTL_Write_lpa	457
15.2.24	qapi_FTL_Erase_block	459
16	Timer	460
16.1	Timer Data Types	461
16.1.1	Data Structure	461
16.1.2	Data Typedef	465
16.1.3	Data Enumeration	466
16.2	API Functions	468
16.2.1	qapi_time_get	468
16.2.2	qapi_Timer_Def	469
16.2.3	qapi_Timer_Set	470
16.2.4	qapi_Timer_Get_Timer_Info	471
16.2.5	qapi_Timer_Sleep	472
16.2.6	qapi_Timer_Undef	473
16.2.7	qapi_Timer_Stop	474

1 DSS Net Control APIs

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

- DSS Netctrl Macros Data Types
- Initialize the DSS Netctrl Library
- Release the DSS Netctrl Library
- Get the Data Service Handle
- Release the Data Service Handle
- Set the Data Call Parameter
- Start a Data Call
- Stop a Data Call
- Get Packet Data Transfer Statistics
- Reset Packet Data Transfer Statistics
- Get the Data Call End Reason
- Get the Data Call Technology
- Get the Data Bearer Technology
- Get the Device Name
- Get the QMI Port Name
- Get the IP Address Count
- Get the IP Address Information
- Get the IP Address Information Structure
- Get the Link MTU Information
- Add Filters for an MO Exception IP Data Call
- Remove Filters for an MO Exception IP Data Call
- Send Non-IP UL Data
- Set Transfer Status
- Get Rate Control

1.1 DSS Netctrl Data Types

This section contains the DSS netctrl constants and macros, enumerations, and data structures.

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

Supported Radio Technologies

- #define QAPI_DSS_RADIO_TECH_MAX 6

Extended Radio Technology

- #define QAPI_DSS_EXT_RADIO_TECH_UNKNOWN 0x00
- #define QAPI_DSS_EXT_RADIO_TECH_MIN 0x01
- #define QAPI_DSS_EXT_RADIO_TECH_NONIP QAPI_DSS_EXT_RADIO_TECH_MIN

Supported Extended Radio Technologies

- #define QAPI_DSS_EXT_RADIO_TECH_MAX 1

MO Exception Data

- #define QAPI_DSS_MO_EXCEPTION_NONE 0x00
- #define QAPI_DSS_MO_EXCEPTION_IP_DATA 0x01



- #define QAPI_DSS_MO_EXCEPTION_NONIP_DATA 0x02

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

Device Name

For example, rmnet_sdioxx, rmnet_xx, etc.

- #define QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN 12

Maximum Client Handles Supported

- #define QAPI_DSS_MAX_DATA_CALLS 20

QAPI_DSS Error Codes

- #define QAPI_DSS_SUCCESS 0
- #define QAPI_DSS_ERROR -1

IP Versions

- #define QAPI_DSS_IP_VERSION_NON_IP 2
- #define QAPI_DSS_IP_VERSION_4 4
- #define QAPI_DSS_IP_VERSION_6 6
- #define QAPI_DSS_IP_VERSION_4_6 10

Supported Modes of Operation

- #define QAPI_DSS_MODE_GENERAL 0

Maximum Supported MO Exception Filters

- #define **QAPI_DSS_MAX_EXCEPTION_FILTERS** 255

Maximum IPv6 Address Length

- #define **QAPI_DSS_IPV6_ADDR_LEN** 16

MO Exception Data Filter Error Mask

- typedef uint64_t **qapi_DSS_MO_Filter_Error_Mask_t**
- #define **QAPI_DSS_FILTER_PARAM_NONE_V01** 0x00000000
- #define **QAPI_DSS_FILTER_PARAM_IP_VERSION_V01** 0x00000001
- #define **QAPI_DSS_FILTER_PARAM_IPV4_SRC_ADDR_V01** 0x00000002
- #define **QAPI_DSS_FILTER_PARAM_IPV4_DEST_ADDR_V01** 0x00000004
- #define **QAPI_DSS_FILTER_PARAM_IPV4_TOS_V01** 0x00000008
- #define **QAPI_DSS_FILTER_PARAM_IPV6_SRC_ADDR_V01** 0x00000010
- #define **QAPI_DSS_FILTER_PARAM_IPV6_DEST_ADDR_V01** 0x00000020
- #define **QAPI_DSS_FILTER_PARAM_IPV6_TRF_CLS_V01** 0x00000040
- #define **QAPI_DSS_FILTER_PARAM_IPV6_FLOW_LABEL_V01** 0x00000080
- #define **QAPI_DSS_FILTER_PARAM_XPORT_PROT_V01** 0x00000100
- #define **QAPI_DSS_FILTER_PARAM_TCP_SRC_PORT_V01** 0x00000200
- #define **QAPI_DSS_FILTER_PARAM_TCP_DEST_PORT_V01** 0x00000400
- #define **QAPI_DSS_FILTER_PARAM_UDP_SRC_PORT_V01** 0x00000800
- #define **QAPI_DSS_FILTER_PARAM_UDP_DEST_PORT_V01** 0x00001000
- #define **QAPI_DSS_FILTER_PARAM_ICMP_TYPE_V01** 0x00002000
- #define **QAPI_DSS_FILTER_PARAM_ICMP_CODE_V01** 0x00004000
- #define **QAPI_DSS_FILTER_PARAM_ESP_SPI_V01** 0x00008000
- #define **QAPI_DSS_FILTER_PARAM_AH_SPI_V01** 0x00010000

MO Exception Data IPv4 Filter Mask

- typedef uint64_t **qapi_DSS_IPv4_Filter_Mask_t**
- #define **QAPI_DSS_IPV4_FILTER_MASK_NONE** 0x00000000



- #define QAPI_DSS_IPV4_FILTER_MASK_SRC_ADDR 0x00000001
- #define QAPI_DSS_IPV4_FILTER_MASK_DEST_ADDR 0x00000002
- #define QAPI_DSS_IPV4_FILTER_MASK_TOS 0x00000004

MO Exception Data IPv6 Filter Mask

- typedef uint64_t qapi_DSS_IPv6_Filter_Mask_t
- #define QAPI_DSS_IPV6_FILTER_MASK_NONE 0x00000000
- #define QAPI_DSS_IPV6_FILTER_MASK_SRC_ADDR 0x00000001
- #define QAPI_DSS_IPV6_FILTER_MASK_DEST_ADDR 0x00000002
- #define QAPI_DSS_IPV6_FILTER_MASK_TRAFFIC_CLASS 0x00000004
- #define QAPI_DSS_IPV6_FILTER_MASK_FLOW_LABEL 0x00000008

Transport Port Filter Mask Information

- typedef uint64_t qapi_DSS_Port_Info_Filter_Mask_t
- #define QAPI_DSS_PORT_INFO_FILTER_MASK_NONE 0x00000000
- #define QAPI_DSS_PORT_INFO_FILTER_MASK_SRC_PORT 0x00000001
- #define QAPI_DSS_PORT_INFO_FILTER_MASK_DEST_PORT 0x00000002

ICMP Filter Mask Information

- typedef uint64_t qapi_DSS_ICMP_Info_Filter_Mask_t
- #define QAPI_DSS_ICMP_FILTER_MASK_NONE 0x00000000
- #define QAPI_DSS_ICMP_FILTER_MASK_MSG_TYPE 0x00000001
- #define QAPI_DSS_ICMP_FILTER_MASK_MSG_CODE 0x00000002

IPSec Filter Mask Information

- typedef uint64_t qapi_DSS_IPSec_Info_Filter_Mask_t
- #define QAPI_DSS_IPSEC_FILTER_MASK_NONE 0x00000000
- #define QAPI_DSS_IPSEC_FILTER_MASK_SPI 0x00000001

Transfer status related

- `#define QAPI_WDS_SET_TRANSFER_STATUS_BUF_SIZE 128`
- `#define QAPI_WDS_STATUS_TRANSFER_MAX_ID 0x7ffffff`

FIBOCOM
Confidential

1.1.1 Data Define

1.1.1.1 #define QAPI_DSS_RADIO_TECH_UNKNOWN 0x00000000

Technology is unknown.

1.1.1.2 #define QAPI_DSS_RADIO_TECH_MIN 0x00000001

Start.

1.1.1.3 #define QAPI_DSS_RADIO_TECH_UMTS QAPI_DSS_RADIO_TECH_MIN

UMTS.

1.1.1.4 #define QAPI_DSS_RADIO_TECH_CDMA 0x00000002

CDMA.

1.1.1.5 #define QAPI_DSS_RADIO_TECH_1X 0x00000004

1X.

1.1.1.6 #define QAPI_DSS_RADIO_TECH_DO 0x00000008

Data only.

1.1.1.7 #define QAPI_DSS_RADIO_TECH_LTE 0x00000010

LTE.

1.1.1.8 #define QAPI_DSS_RADIO_TECH_TDSCDMA 0x00000020

TDSCDMA.

1.1.1.9 #define QAPI_DSS_MO_EXCEPTION_NONE 0x00

None.

1.1.1.10 #define QAPI_DSS_MO_EXCEPTION_IP_DATA 0x01

MO exception IP data.

1.1.1.11 #define QAPI_DSS_MO_EXCEPTION_NONIP_DATA 0x02

MO exception non-IP data.

1.1.1.12 #define QAPI_DSS_CALL_INFO_USERNAME_MAX_LEN 127

Maximum length of the username.

1.1.1.13 #define QAPI_DSS_CALL_INFO_PASSWORD_MAX_LEN 127

Maximum length of the password.

1.1.1.14 #define QAPI_DSS_CALL_INFO_APN_MAX_LEN 150

Maximum length of the APN.

1.1.1.15 #define QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN 12

Maximum length of the device name.

1.1.1.16 #define QAPI_DSS_SUCCESS 0

Indicates that the operation was successful.

1.1.1.17 #define QAPI_DSS_ERROR -1

Indicates that the operation was not successful.

1.1.1.18 #define QAPI_DSS_IP_VERSION_NON_IP 2

IP version Non IP.

1.1.1.19 #define QAPI_DSS_IP_VERSION_4 4

IP version v4.

1.1.1.20 #define QAPI_DSS_IP_VERSION_6 6

IP version v6.

1.1.1.21 #define QAPI_DSS_IP_VERSION_4_6 10

IP version v4v6.

1.1.1.22 #define QAPI_DSS_TRANSFER_STATUS_NO_INFO_AVAIL 0x00

Subsequent uplink and downlink data transfer information. No information available.

1.1.1.23 #define QAPI_DSS_TRANSFER_STATUS_NO_UPLINK_DOWNLINK_DATA 0x01

No further uplink or downlink data transmission.

1.1.1.24 #define QAPI_DSS_TRANSFER_STATUS_SINGLE_DOWNLINK_DATA 0x02

Only a single downlink transmission and no further uplink data.

1.1.1.25 #define QAPI_DSS_FILTER_PARAM_NONE_V01 0x00000000

No errors.

1.1.1.26 #define QAPI_DSS_FILTER_PARAM_IP_VERSION_V01 0x00000001

IP version.

1.1.1.27 #define QAPI_DSS_FILTER_PARAM_IPV4_SRC_ADDR_V01 0x00000002

IPv4 source address.

1.1.1.28 #define QAPI_DSS_FILTER_PARAM_IPV4_DEST_ADDR_V01 0x00000004

IPv4 destination address.

1.1.1.29 #define QAPI_DSS_FILTER_PARAM_IPV4_TOS_V01 0x00000008

1.1.1.30 #define QAPI_DSS_FILTER_PARAM_IPV6_SRC_ADDR_V01 0x00000010

IPv6 source address.

1.1.1.31 #define QAPI_DSS_FILTER_PARAM_IPV6_DEST_ADDR_V01 0x00000020

IPv6 destination address.

1.1.1.32 #define QAPI_DSS_FILTER_PARAM_IPV6_TRF_CLS_V01 0x00000040

IPv6 traffic class.

1.1.1.33 #define QAPI_DSS_FILTER_PARAM_IPV6_FLOW_LABEL_V01 0x00000080

IPv6 flow label.

1.1.1.34 #define QAPI_DSS_FILTER_PARAM_XPORT_PROT_V01 0x00000100

Transport protocol.

1.1.1.35 #define QAPI_DSS_FILTER_PARAM_TCP_SRC_PORT_V01 0x00000200

TCP source port.

1.1.1.36 #define QAPI_DSS_FILTER_PARAM_TCP_DEST_PORT_V01 0x00000400

TCP destination port.

1.1.1.37 #define QAPI_DSS_FILTER_PARAM_UDP_SRC_PORT_V01 0x00000800

UDP source port.

1.1.1.38 #define QAPI_DSS_FILTER_PARAM_UDP_DEST_PORT_V01 0x00001000

UDP destination port.

1.1.1.39 #define QAPI_DSS_FILTER_PARAM_ICMP_TYPE_V01 0x00002000

ICMP type.

1.1.1.40 #define QAPI_DSS_FILTER_PARAM_ICMP_CODE_V01 0x00004000

ICMP code.

1.1.1.41 #define QAPI_DSS_FILTER_PARAM_ESP_SPI_V01 0x00008000

Encapsulating SPI.

1.1.1.42 #define QAPI_DSS_FILTER_PARAM_AH_SPI_V01 0x00010000

Authentication header SPI.

1.1.1.43 #define QAPI_DSS_IPV4_FILTER_MASK_NONE 0x00000000

No parameters.

1.1.1.44 #define QAPI_DSS_IPV4_FILTER_MASK_SRC_ADDR 0x00000001

IPv4 source address.

1.1.1.45 #define QAPI_DSS_IPV4_FILTER_MASK_DEST_ADDR 0x00000002

IPv4 destination address.

1.1.1.46 #define QAPI_DSS_IPV4_FILTER_MASK_TOS 0x00000004

IPv4 traffic class.

1.1.1.47 #define QAPI_DSS_IPV6_FILTER_MASK_NONE 0x00000000

No parameters.

1.1.1.48 #define QAPI_DSS_IPV6_FILTER_MASK_SRC_ADDR 0x00000001

IPv6 source address.

1.1.1.49 #define QAPI_DSS_IPV6_FILTER_MASK_DEST_ADDR 0x00000002

IPv6 destination address.

1.1.1.50 #define QAPI_DSS_IPV6_FILTER_MASK_TRAFFIC_CLASS 0x00000004

IPv6 traffic class.

1.1.1.51 #define QAPI_DSS_IPV6_FILTER_MASK_FLOW_LABEL 0x00000008

IPv6 flow label.

1.1.1.52 #define QAPI_DSS_PORT_INFO_FILTER_MASK_NONE 0x00000000

No parameters.

1.1.1.53 #define QAPI_DSS_PORT_INFO_FILTER_MASK_SRC_PORT 0x00000001

Source port.

1.1.1.54 #define QAPI_DSS_PORT_INFO_FILTER_MASK_DEST_PORT 0x00000002

Destination port.

1.1.1.55 #define QAPI_DSS_ICMP_FILTER_MASK_NONE 0x00000000

No parameters.

1.1.1.56 #define QAPI_DSS_ICMP_FILTER_MASK_MSG_TYPE 0x00000001

Message type.

1.1.1.57 #define QAPI_DSS_ICMP_FILTER_MASK_MSG_CODE 0x00000002

Message code.

1.1.1.58 #define QAPI_DSS_IPSEC_FILTER_MASK_NONE 0x00000000

1.1.1.59 #define QAPI_DSS_IPSEC_FILTER_MASK_SPI 0x00000001

Security parameter index.

1.1.1.60 #define QAPI_WDS_SET_TRANSFER_STATUS_BUF_SIZE 128

Transfers status buffer size.

1.1.1.61 #define QAPI_WDS_STATUS_TRANSFER_MAX_ID 0x7fffffff

Transfers status maximum ID.

1.1.1.62 #define qapi_DSS_Release(a) dss_destroy_indirection(a, TXM_QAPI_DSS-_RELEASE)

Macro that releases Byte Pool Pointer for DSS Application. Parameter 'a': Handle.

On success, QAPI_OK is returned. On error, QAPI_ERROR is returned.



Note:

This macro is only used in DAM Space.

1.1.1.63 #define qapi_DSS_Pass_Pool_Ptr(a, b) dss_set_byte_pool(a,b)

Macro that passes Byte Pool Pointer for DSS Application. Parameter 'a': Handle.

Parameter 'b': Pointer to Byte Pool.

On success, QAPI_OK is returned. On error, QAPI_ERROR is returned.



Note:

This macro is only used in DAM Space.

1.1.2 Data Structure

1.1.2.1 struct qapi_DSS_CE_Reason_s

Call end (CE) reason.

Data fields

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.

1.1.2.2 struct qapi_DSS_Call_Param_Value_s

Specifies call parameter values.

Data fields

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.

1.1.2.3 struct qapi_DSS_Addr_s

Structure to represent the IP address.

Data fields

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.

1.1.2.4 union qapi_DSS_Addr_s::qapi_dss_ip_address_u

Union of DSS IP addresses.

Data fields

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.

1.1.2.5 struct qapi_DSS_Addr_Info_s

IP address-related information.

Data fields

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.

1.1.2.6 struct qapi_DSS_Data_Pkt_Stats_s

Packet statistics.

Data fields

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.

1.1.2.7 struct qapi_DSS_Evt_Payload_s

Event payload sent with event callbacks.

Data fields

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

1.1.2.8 struct qapi_DSS_IPv4_Filter_Address_Type_s

IPv4 address filter type.

Data fields

Type	Parameter	Description
uint32_t	ipv4_addr	IPv4 address.
uint32_t	subnet_mask	IPv4 subnet mask.

1.1.2.9 struct qapi_DSS_IPv4_Filter_TOS_Type_s

IPv4 TOS filter type.

Data fields

Type	Parameter	Description
uint8_t	val	Type of service value.
uint8_t	mask	Type of service mask.

1.1.2.10 struct qapi_DSS_IPv4_Filter_Info_s

IPv4 filter rule information.

Data fields

Type	Parameter	Description
qapi_DSS_IPv4_Filter_Mask_t	valid_params	Bitmask that denotes which parameters contain valid values.

qapi_DSS_IPv4_Filter_Address_Type_t	src_addr	IPv4 source address.
qapi_DSS_IPv4_Filter_Address_Type_t	dest_addr	IPv4 destination address.
qapi_DSS_IPv4_Filter_TOS_Type_t	tos	IPv4 type of service.

1.1.2.11 struct qapi_DSS_IPv6_Filter_Address_Type_s

IPv6 address filter type.

Data fields

Type	Parameter	Description
uint8_t	ipv6_address	IPv6 address.
uint8_t	prefix_len	IPv6 address prefix length.

1.1.2.12 struct qapi_DSS_IPv6_Filter_Traffic_Type_s

IPv6 traffic class filter type.

Data fields

Type	Parameter	Description
uint8_t	val	Traffic class value.
uint8_t	mask	Traffic class mask.

1.1.2.13 struct qapi_DSS_IPv6_Filter_Info_s

IPv6 filter rule information.

Data fields

Type	Parameter	Description
qapi_DSS_IPv6_Filter_Info_s.valid_params	valid_params	Bitmask that denotes which parameters contain valid values.
qapi_DSS_IPv6_Filter_Info_s.src_addr	src_addr	IPv6 source address.
qapi_DSS_IPv6_Filter_Info_s.dest_addr	dest_addr	IPv6 destination address.
qapi_DSS_IPv6_Filter_Info_s.trf_cls	trf_cls	IPv6 traffic class.
uint32_t	flow_label	IPv6 flow label.

1.1.2.14 struct qapi_DSS_IP_Header_Filters_s

Internet protocol filter rule parameters.

Data fields

Type	Parameter	Description
------	-----------	-------------

uint8_t	ip_version	Depending on the IP version set, either the IPv4 or the IPv6 information is valid. Values: QAPI_DSS_IP_VERSION_4 (0x04) – IPv4 QAPI_DSS_IP_VERSION_6 (0x06) – IPv6
qapi_DSS_IPv4_Filter_Info_t	v4_info	Filter parameters for IPv4.
qapi_DSS_IPv6_Filter_Info_t	v6_info	Filter parameters for IPv6.

1.1.2.15 struct qapi_DSS_Port_Type_s

DSS port type.

Data fields

Type	Parameter	Description
uint16_t	port	Port.
uint16_t	range	Range.

1.1.2.16 struct qapi_DSS_Port_Filter_Info_s

TCP and UDP port filter rule parameters.

Data fields

Type	Parameter	Description
qapi_DSS_	valid_params	Bitmask that denotes which parameters contain valid values.

Port_Info_- Filter_Mask_t		
qapi_DSS_- Port_Type_t	src_port_info	Source port information.
qapi_DSS_- Port_Type_t	dest_port_info	Destination port information.

1.1.2.17 struct qapi_DSS_ICMP_Info_Filter_Type_s

ICMP filter rule parameters.

Data fields

Type	Parameter	Description
qapi_DSS_I- CMP_Info_- Filter_Mask_t	valid_params	Bitmask that denotes which parameters contain valid values.
uint8_t	type	ICMP type.
uint8_t	code	ICMP code.

1.1.2.18 struct qapi_DSS_IPSec_Info_Filter_Type_s

IPSec filter rule parameters.

Data fields

Type	Parameter	Description
qapi_DSS_I-	valid_params	Bitmask that denotes which parameters contain valid values.

PSec_Info_- Filter_Mask_t		
uint32_t	spi	Security parameter index for IPSec.

1.1.2.19 struct qapi_DSS_Xport_Header_Filters_s

Transport protocol filter rule parameters.

Data fields

Type	Parameter	Description
qapi_DSS_XPORT_Protocol- _t	xport_protocol	Depending on the value in xport_protocol, only one field of icmp_info, tcp_info, udp_info, esp_info, or ah_info is valid. QAPI_DSS_XPORT_PROTO_NONE implies that no transport level protocol parameters are valid.
qapi_DSS_Port_Filter_- Info_t	tcp_info	Filter parameters for TCP.
qapi_DSS_Port_Filter_- Info_t	udp_info	Filter parameters for UDP.
qapi_DSS_ICMP_Info_- Filter_Type_t	icmp_info	Filter parameters for ICMP.
qapi_DSS_IPSec_Info_- Filter_Type_t	esp_info	Filter parameters for ESP.
qapi_DSS_IPSec_Info_- Filter_Type_t	ah_info	Filter parameters for AH.

PSec_Info_- Filter_Type_t		
------------------------------	--	--

1.1.2.20 struct qapi_DSS_Filter_Rule_Type_s

MO exception data filter rules.

Data fields

Type	Parameter	Description
qapi_DSS_IP_- Header_Filters- _t	ip_info	Internet protocol filter parameters.
qapi_DSS_- Xport_Header- _Filters_t	xport_info	Transport level protocol filter parameters.

1.1.2.21 struct qapi_DSS_Add_MO_Exception_Filters_Req_s

Add an MO exception data filters request.

Data fields

Type	Parameter	Description
uint8_t	filter_rules_- valid	Set to TRUE if filter rules are being passed.
uint32_t	filter_rules_len	Set to the number of elements in the filter rules.
qapi_DSS_-	filter_rules	List of filter rules.

Filter_Rule_- Type_t		
-------------------------	--	--

1.1.2.22 struct qapi_DSS_Add_MO_Exception_Filters_Rsp_s

Add an MO exception data filters response.

Data fields

Type	Parameter	Description
uint8_t	filter_handles_- valid	Set to TRUE if filter handles are being passed. Filter handles will be passed if atleast one of the filter rules got added.
uint32_t	filter_handles_- len	Set to the number of filter rules that got added.
uint32_t	filter_handles	List of handles that uniquely identify added filter rules. Filter handles will be present at indices corresponding to filter rules.
uint8_t	filter_rule_- error_valid	Set to TRUE if any filter rule errors are being passed. Filter errors will be passed if any of the filter rules didn't got added.
uint32_t	filter_rule_- error_len	Set to the number of filter rules that didn't got added.
qapi_DSS_M- O_Filter_Error- _Mask_t	filter_rule_error	Error mask list for filter rule errors. Filter errors will be present at indices corresponding to filter rules.

1.1.2.23 struct qapi_DSS_Apn_Rate_Control_Info_s

APN rate control information.

Type	Parameter	Description
uint32_t	uplink_time_unit	Represents the time unit that the uplink data is restricted.
uint32_t	max_uplink_rate	Maximum number of the uplink packets that can be sent within the uplink time limit.

1.1.2.24 struct qapi_DSS_Remove_MO_Exceptional_Filters_s

Remove MO exception data filters.

Data fields

Type	Parameter	Description
uint32_t	filter_handles_len	Set to the number of elements in the filter handles.
uint32_t	filter_handles	List of handles to the filter rules to remove.

1.1.2.25 union .u

Union of values.

Data fields

Type	Parameter	Description
u	valuebuf	Union of buffer values.
int64_t	valueint	Requests integer value.
bool	valuebool	Requests Boolean value.

double	valuedouble	Requests double values .
--------	-------------	--------------------------

1.1.2.26 struct .u.valuebuf

Union of buffer values.

Data fields

Type	Parameter	Description
char	buf	Request buffer.
uint32_t	len	Length of the request string.

1.1.3 Data Typedef

1.1.3.1 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.

Parameters

in	<i>hndl</i>	Handle to which this event is associated.
in	<i>user_data</i>	Application-provided user data.
in	<i>evt</i>	Event identifier.
in	<i>payload_ptr</i>	Pointer to associated event information.

Returns

None.

1.1.4 Data Enumeration

1.1.4.1 enum qapi_DSS_Auth_Pref_e

Authentication preference for a PDP connection.

Enumerator:

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

1.1.4.2 enum qapi_DSS_CE_Reason_Type_e

Call end reason type.

Enumerator:

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

1.1.4.3 enum qapi_DSS_Call_Info_Enum_e

Call parameter identifier.

Enumerator:

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.
QAPI_DSS_CALL_INFO_MAX_E	

1.1.4.4 enum qapi_DSS_Net_Evt_e

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

Enumerator:

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.

1.1.4.5 enum qapi_DSS_IP_Family_e

IP families.

Enumerator:

QAPI_DSS_IP_FAMILY_V4_E	IPV4 address family
QAPI_DSS_IP_FAMILY_V6_E	IPV6 address family

1.1.4.6 QAPI_DSS_NUM_IP_FAMILIES_E enum qapi_DSS_Data_Bearer_Tech_e

Bearer technology types.

Enumerator:

QAPI_DSS_DATA_BEARER_TECH_UNKNOWN_E	Unknown bearer.
QAPI_DSS_DATA_BEARER_TECH_FMC_E	Fixed mobile convergence.
QAPI_DSS_DATA_BEARER_TECH_GPRS_E	GPRS.
QAPI_DSS_DATA_BEARER_TECH_LTE_E	LTE.
QAPI_DSS_DATA_BEARER_TECH_GSM_E	GSM.

1.1.4.7 enum qapi_DSS_Call_Tech_Type_e

Call technology.

Enumerator:

QAPI_DSS_CALL_TECH_INVALID_E	Invalid technology.
QAPI_DSS_CALL_TECH_CDMA_E	CDMA.
QAPI_DSS_CALL_TECH_UMTS_E	UMTS.

1.1.4.8 enum qapi_DSS_XPORT_Protocol_e

MO exception data transport protocol information.

Enumerator:

QAPI_DSS_XPORT_PROTO_NONE	No transport protocol
QAPI_DSS_XPORT_PROTO_ICMP	Internet Control Messaging Protocol
QAPI_DSS_XPORT_PROTO_TCP	Transmission Control Protocol
QAPI_DSS_XPORT_PROTO_UDP	User Datagram Protocol
QAPI_DSS_XPORT_PROTO_ESP	Encapsulating Security Payload protocol
QAPI_DSS_XPORT_PROTO_AH	Authentication Header Protocol.
QAPI_DSS_XPORT_PROTO_ICMP6	ICMPv6 Protocol.



QAPI_DSS_XPORT_PROTO_TCPUDP
requests.

TCP and UDP protocol; only applicable for remote socket

Transfer status information types.

Enumerator:

QAPI_SET_STATUS_INFO_TYPE_BOOLEAN_E *Request type is Boolean.*

QAPI_SET_STATUS_TYPE_INTEGER_E *Request type is integer.*

QAPI_SET_STATUS_INFO_TYPE_BUFFER_E *Request type is buffer.*

QAPI_SET_STATUS_INFO_TYPE_DOUBLE_E *Request type is array*

Transfer status ID.

Enumerator:

QAPI_SET_TRANSFER_STATUS_INVALID *Invalid ID.*

QAPI_SET_TRANSFER_STATUS_RAI *Status ID for RAI Indication.*

QAPI_SET_NEXT_ACTIVITY_TIMER *Status ID for Next Activity Timer.*

1.2 API Functions

1.2.1 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.

Prototype

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



Note:

Only QAPI_DSS_MODE_GENERAL is to be used by applications.

Parameters

in	mode	Mode of operation in which to initialize the library.
----	------	---

Returns

QAPI_OK – Initialization was successful. QAPI_ERROR – Initialization failed.

Dependencies

None.

1.2.2 qapi_DSS_Release

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

Prototype

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



Note:

Only QAPI_DSS_MODE_GENERAL is to be used by applications.

Parameters

in	mode	Mode of operation in which to de-initialize the library.
----	------	--

Returns

QAPI_OK – Clean up was successful.

QAPI_ERROR – Cleanup failed.

Dependencies

None.

1.2.3 qapi_DSS_Get_Data_Srvc_Hndl

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

Prototype

```
qapi_Status_t qapi_DSS_Get_Data_Srvc_Hndl (qapi_DSS_Net_Ev_CB_tuser_cb_fn, void * user_data,
qapi_DSS_Hndl_t * hndl)
```



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.

Any events to the application are forwarded to through the registered application callback. When the application is running in Kernel mode, the resources allocated for the event payload need to be released by the application. On the other hand, if the application is running in user space the resources are automatically released when the callback returns.

Parameters

in	user_cb_fn	Client callback function used to post event indications.
in	user_data	Pointer to the client context block (cookie). The value may be NULL.
in	hndl	Pointer to data service handle.

Returns

QAPI_OK – Operation was successful. QAPI_ERROR – Operation failed.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

1.2.4 qapi_DSS_Rel_Data_Srvc_Hndl

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

Prototype

qapi_Status_t qapi_DSS_Rel_Data_Srvc_Hndl (qapi_DSS_Hndl_t hndl)



Note:

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

Parameters

in	hndl	Handle received from qapi_DSS_Get_Data_Srvc_Hndl() .
----	------	--

Returns

QAPI_OK – Operation was successful. QAPI_ERROR – Operation failed.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_Hndl\(\)](#).

1.2.5 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

in	hndl	Handle received from qapi_DSS_Get_Data_Srvc_Hndl() .
in	identifier	Identifies the parameter information.
in	info	Parameter value that is to be set.

Returns

QAPI_OK – Data call parameter was set successfully. QAPI_ERROR – Data call parameter was not set successfully.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_Hndl\(\)](#).

1.2.6 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

in	hndl	Handle received from qapi_DSS_Get_Data_Srvc_Hndl() .
----	------	--

Returns

QAPI_OK – Data call start request was sent successfully. QAPI_ERROR – Data call start request was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_Hndl\(\)](#).

1.2.7 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_Svc_Hndl\(\)](#) indicating the data call tear-down status.

Prototype

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

Parameters

in	hndl	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
----	------	---

Returns

QAPI_OK – Data call stop request was sent successfully.

QAPI_ERROR – Data call stop request was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

The data call must have been brought up using [qapi_DSS_Start_Data_Call\(\)](#).

1.2.8 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

in	hndl	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
in	dss_data_stats	Buffer to hold the queried statistics details.

Returns

QAPI_OK – Packet data transfer statistics were queried successfully.

QAPI_ERROR – Packet data transfer statistics query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.9 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

in	hndl	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
----	------	---

Returns

QAPI_OK – Packet data transfer statistics were reset successfully. QAPI_ERROR – Packet data transfer statistics reset was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_Hndl\(\)](#).

1.2.10qapi_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

in	hndl	Handle received from qapi_DSS_Get_Data_Srv_Hndl() .
out	ce_reason	Buffer to hold data call ending reason information.
in	ip_family	IP family for which the call end reason was requested.

Returns

QAPI_OK – Data call end reason was queried successfully.

QAPI_ERROR – Data call end reason query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srv_Hndl\(\)](#).

1.2.11 qapi_DSS_Get_Call_Tech

Gets the technology 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

in	hndl	Handle received from qapi_DSS_Get_Data_Srv_Hndl() .
out	call_tech	Buffer to hold the call technology

Returns

QAPI_OK – Data call bring-up technology was queried successfully.

QAPI_ERROR – Data call bring-up technology query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srv_Hndl\(\)](#).

1.2.12qapi_DSS_Get_Current_Data_Bearer_Tech

Queries the data bearer technology 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

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
in	<i>bearer_tech</i>	Pointer to where to retrieve the data bearer technology.

Returns

QAPI_OK – Data bearer technology was returned successfully.

QAPI_ERROR – Data bearer technology was not returned successfully.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.13 qapi_DSS_Get_Device_Name

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

Prototype

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



Note:

len must be at least QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN + 1 long.

Parameters

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
out	<i>buf</i>	Buffer to hold the data interface name string.
in	<i>len</i>	Length of the buffer allocated by the client.

Returns

QAPI_OK – Data interface name was returned successfully.

QAPI_ERROR – Data interface name was not returned successfully.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.14qapi_DSS_Get_Qmi_Port_Name

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

Prototype

qapi_Status_t qapi_DSS_Get_Qmi_Port_Name (qapi_DSS_Hndl_t hndl, char buf, int len)



Note:

len must be at least DSI_CALL_INFO_DEVICE_NAME_MAX_LEN + 1 long.

Parameters

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
out	<i>buf</i>	Buffer to hold the QMI port name string.
in	<i>len</i>	Length of the buffer allocated by the client.

Returns

QAPI_OK – Port name was returned successfully.

QAPI_ERROR – Port name was not returned successfully.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.15 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

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
in	<i>ip_addr_cnt</i>	Pointer to where to retrieve the number of IP addresses associated with the DSS interface.

Returns

QAPI_OK – IP address count query was successful.

QAPI_ERROR – IP address count query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.16qapi_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

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
out	<i>info_ptr</i>	Buffer containing the IP address information.
in	<i>len</i>	Number of IP address buffers

Returns

QAPI_OK – IP address query was successful.

QAPI_ERROR – IP address query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_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.

1.2.17 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

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
out	<i>info_ptr</i>	Buffer containing the IP address information.
in	<i>addr_family</i>	IPv4/IPv6.

Returns

QAPI_OK – IP address query was successful.

QAPI_ERROR – IP address query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_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

1.2.18qapi_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

in	hndl	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
out	mtu	Buffer containing the MTU information. For an IPv4v6 call, if IPv4 and IPv6 have different MTU sizes, the smaller one will be returned.

Returns

QAPI_OK – MTU query was successful.

QAPI_ERROR – MTU query was unsuccessful.

Dependencies

qapi_DSS_Init() must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.19 qapi_DSS_Get_Link_Mtu_per_family

Queries the MTU information associated with the link per_family.

Prototype

```
qapi_Status_t qapi_DSS_Get_Link_Mtu_per_family ( qapi_DSS_Hndl_t netmgr_hndl, unsigned int
ip_family, unsigned int * mtu )
```

Parameters

in	netmgr_hndl	Handle received from qapi_DSS_Get_Data_Srvc_Hndl() .
in	ip_family	Value defined by qapi_DSS_IP_Family_e .
out	mtu	Buffer containing the MTU information.

Returns

QAPI_OK – MTU query was successful.

QAPI_ERROR – MTU query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_Hndl\(\)](#).

1.2.20qapi_DSS_Add_MO_Exception_IPdata_Filters

Adds filters for an MO exception IP data call.

Prototype

```
qapi_Status_t qapi_DSS_Add_MO_Exception_IPdata_Filters ( qapi_DSS_Hndl_t hndl,
qapi_DSS_Add_MO_Exception_Filters_Req_t * filter_req,qapi_DSS_Add_MO_Exception_Filters_Rsp_t
* filter_rsp )
```

Parameters

in	hndl	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
in	filter_req	Filter rules information to be added.
out	filter_rsp	Filter rules handles and error information.

Returns

QAPI_OK – Adding filter rules will be successful if atleast one of the filter requests get added.

QAPI_ERROR – Adding filter rules was unsuccessful. None of the filter request got added.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.21 qapi_DSS_Remove_MO_Exception_IPdata_Filters

Removes filters for an MO exception IP data call.

Prototype

```
qapi_Status_t qapi_DSS_Remove_MO_Exception_IPdata_Filters ( qapi_DSS_Hndl_t hndl,
qapi_DSS_Remove_MO_Exception_Filters_t * filter_req )
```

Parameters

in	hndl	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
in	filter_req	Filter rules information to be removed.

Returns

QAPI_OK – Removing filter rules was successful.

QAPI_ERROR – Removing filter rules was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.22qapi_DSS_Nipd_Send

Sends non-IP UL data. In the DL, non-IP data received by the DSS module is passed to the application using the registered application callback.

Prototype

```
qapi_Status_t qapi_DSS_Nipd_Send ( qapi_DSS_Hndl_t hndl, uint8_t * data,uint32_t data_len, uint8_t ex_data )
```

Parameters

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Srvc_Hndl() .
in	<i>data</i>	Non-IP data payload buffer that is to be sent.
in	<i>data_len</i>	Length of the data payload to be sent.
in	<i>ex_data</i>	MO exception, non-IP or not: QAPI_DSS_MO_EXCEPTION_NONIP_DATA or QAPI_DSS_MO_EXCEPTION_NONE.

Returns

QAPI_OK – Send Data was successful.

QAPI_ERROR – Send Data was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_Hndl\(\)](#).

1.2.23qapi_DSS_Set_Transfer_Status_v2

Sets the data transmission status or next activity timer.

Prototype

```
qapi_Status_t qapi_DSS_Set_Transfer_Status_v2 ( qapi_DSS_Hndl_t netmgr_hndl,  
qapi_Set_transfer_Status_Info_t * status_info, int num_of_elements)
```

Parameters

in	<i>netmgr_hndl</i>	Handle received from qapi_DSS_Get_Data_Srvc_Hndl() .
in	<i>status_info</i>	Contains information to set RAI or NAT.
in	<i>num_of_elements</i>	Total RAI or NAT elements for status information to set .

Returns

QAPI_OK – Data transmission status was set successfully.

QAPI_ERROR – Data transmission status was not set successfully.

Dependencies

[qapi_DSS_Init\(\)](#) must be called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_Hndl\(\)](#).

QAPI_Information: If user sets NAT, it is mandatory to set RAI . Only setting NAT will return error .

1.2.24qapi_DSS_Get_Apn_Rate_Control

Gets APN rate control information.

Prototype

```
qapi_Status_t qapi_DSS_Get_Apn_Rate_Control ( qapi_DSS_Hndl_t  
hndl, qapi_DSS_Apn_Rate_Control_Info_t * apn_rate_control )
```

Parameters

in	<i>hndl</i>	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
out	<i>apn_rate_control</i>	APN rate control information.

Returns

QAPI_OK – APN rate control query was successful.

QAPI_ERR_NOT_SUPPORTED - APN rate control is not configured from Network.

QAPI_ERROR – APN rate control query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must be called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#).

1.2.25qapi_DSS_Get_Splmn_Rate_Control

Gets SPLMN rate control information.

Prototype

```
qapi_Status_t qapi_DSS_Get_Splmn_Rate_Control ( qapi_DSS_Hndl_t hndl,uint32_t *
splmn_rate_control )
```

Parameters

in	hndl	Handle received from qapi_DSS_Get_Data_Svc_Hndl() .
out	splmn_rate_control	SPLMN rate control information.

Returns

QAPI_OK – SPLMN rate control query was successful.

QAPI_ERROR – SPLMN rate control query was unsuccessful.

Dependencies

[qapi_DSS_Init\(\)](#) must be called first.

A valid handle must be obtained by [qapi_DSS_Get_Data_Svc_Hndl\(\)](#)

2 QAPI Networking Socket

The QAPI networking 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.

- [QAPI Data Types](#)
- [Create a Socket](#)
- [Bind a Socket](#)
- [Make a Socket Passive](#)
- [Accept a Socket Connection Request](#)
- [Connect to a Socket](#)
- [Set Socket Options](#)
- [Get Socket Options](#)
- [Close a Socket](#)
- [Get a Socket Error Code](#)
- [Receive a Message from a Socket](#)
- [Receive a Message from a Connected Socket](#)
- [Send a Message on a Socket](#)
- [Send a Message on a Connected Socket](#)
- [Select a Socket](#)
- [Initialize a Socket](#)
- [Clear a Socket from a Socket Set](#)
- [Add a Socket to a Socket Set](#)



- Check Whether a Socket is in a Socket Set
- Get the Address of a Connected Peer
- Get the Address to Which the Socket is Bound

FIBOCOM
Confidential

2.1 QAPI Data Types

This section provides the QAPI socket macros and data structures.

The BSD socket interface API is a collection of standard functions that allow the application to include internet communications capabilities. 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 and Client modes
- 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 check for return values of each function.

There is a chance that an operation may fail because of resource limitations. For example, the send function might be able to only send some data, but not all of it in a single call. A subsequent call with the rest of the data is required.

In other cases, an application thread might need to sleep to allow the system to clear its queues, process data, etc.

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 EMSGSIZE 12
- #define EPIPE 13
- #define EDESTADDRREQ 14



- #define ESHUTDOWN 15
- #define ENOPROTOOPT 16
- #define EHAVEOOB 17
- #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

Socket Options

- #define SOL_SOCKET -1
- #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_WINSIZE 0x00400
- #define SO_TIMESTAMP 0x00800
- #define SO_BIGCWND 0x01000



- #define SO_HDRINCL 0x02000
- #define SO_NOSLOWSTART 0x04000
- #define SO_FULLMSS 0x08000
- #define SO_NONIP_DSSHNDL 0x10000
- #define SO_NONIP_EXTYPE 0x20000
- #define SO_SNDBUF 0x1001
- #define SO_RCVBUF 0x1002
- #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 SO_TCP_ACKDELAYTIME 0x2001
- #define SO_TCP_NOACKDELAY 0x2002
- #define TCP_MAXSEG 0x2003
- #define TCP_NODELAY 0x2004
- #define TCP_MAX_RXT 0x2005
- #define TCP_OT 0x2006
- #define TCP_IRT 0x2007
- #define TCP_KEEPIPLE 0x2008
- #define TCP_KEEPIPLEVL 0x2009
- #define TCP_KEEPCNT 0x2010
- #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_OPTIONS 1
- #define IP_TOS 3
- #define IP_TTL_OPT 4
- #define IPV6_SCOPEID 14
- #define IPV6_UNICAST_HOPS 15
- #define IPV6_TCLASS 16

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

Flags for qapi_shutdown()

- #define SHUT_RD 0
- #define SHUT_WR 1
- #define SHUT_RDWR 2

Address is already in use.

FIBOCOM
Confidential

2.1.1 Data Define

2.1.1.1 #define AF_UNSPEC 0

Address family is unspecified.

2.1.1.2 #define AF_INET 2

Address family is IPv4.

2.1.1.3 #define AF_INET6 3

Address family is IPv6.

2.1.1.4 #define AF_INET_DUAL46 4

Address family is IPv4 and IPv6.

2.1.1.5 #define AF_NONIP 11

Address family is NONIP.

2.1.1.6 #define SOCK_STREAM 1

Socket stream (TCP).

2.1.1.7 #define SOCK_DGRAM 2

Socket datagram (UDP).



2.1.1.8 #define SOCK_RAW 3

Raw socket.

2.1.1.9 #define SOCK_NONIP 4

NonIP socket.

2.1.1.10 #define ENOBUFS 1

No buffer space is available.

2.1.1.11 #define ETIMEDOUT 2

Operation timed out.

2.1.1.12 #define EISCONN 3

Socket is already connected.

2.1.1.13 #define EOPNOTSUPP 4

Operation is not supported.

2.1.1.14 #define ECONNABORTED 5

Software caused a connection abort.

2.1.1.15 #define EWOULDBLOCK 6

Socket is marked nonblocking and the requested operation will block.

2.1.1.16 #define ECONNREFUSED 7

Connection was refused.

2.1.1.17 #define ECONNRESET 8

Connection was reset by peer.

2.1.1.18 #define EBADF 9

An invalid descriptor was specified.

2.1.1.19 #define EALREADY 10

Operation is already in progress.

2.1.1.20 #define EMSGSIZE 12

Message is too long.

2.1.1.21 #define EPIPE 13

The local end has been shut down on a connection-oriented socket.

2.1.1.22 #define EDESTADDRREQ 14

2.1.1.23 #define ESHUTDOWN 15

Cannot send after a socket shutdown.

2.1.1.24 #define ENOPROTOOPT 16

Protocol is not available.

2.1.1.25 #define EHAVEOOB 17

Out of band.

2.1.1.26 #define EADDRNOTAVAIL 19

Cannot assign the requested address.

2.1.1.27 #define EAFNOSUPPORT 21

Address family is not supported by the protocol family.

2.1.1.28 #define EINPROGRESS 22

Operation is in progress.

2.1.1.29 #define ELOWER 23

Lower layer (IP) error.

2.1.1.30 #define ENOTSOCK 24

Socket operation on nonsocket.

2.1.1.31 #define EIEIO 27

I/O error.

2.1.1.32 #define ETOOMANYREFS 28

Too many references.

2.1.1.33 #define EFAULT 29

Bad address.

2.1.1.34 #define ENETUNREACH 30

Network is unreachable.

2.1.1.35 #define ENOTCONN 31

Socket is not connected.

2.1.1.36 #define SOL_SOCKET -1

For use with [gs]etsockopt() at the socket level.

2.1.1.37 #define SOL_SOCKET -1

For use with [gs]etsockopt() at the socket level.

2.1.1.38 #define SO_ACCEPTCONN 0x00002

Socket has had listen().

2.1.1.39 #define SO_REUSEADDR 0x00004

Allow local address reuse.

2.1.1.40 #define SO_KEEPALIVE 0x00008

Keep connections alive.

2.1.1.41 #define SO_DONTROUTE 0x00010

Not used.

2.1.1.42 #define SO_BROADCAST 0x00020

Not used.

2.1.1.43 #define SO_USELOOPBACK 0x00040

Not used.

2.1.1.44 #define SO_LINGER 0x00080

2.1.1.45 #define SO_OOINLINE 0x00100

Leave the received OOB data in line.

2.1.1.46 #define SO_TCPACK 0x00200

Allow TCP SACK (selective acknowledgment).

2.1.1.47 #define SO_WINSIZE 0x00400

Set the scaling window option.

2.1.1.48 #define SO_TIMESTAMP 0x00800

Set the TCP timestamp option.

2.1.1.49 #define SO_BICWND 0x01000

Large initial TCP congestion window.

2.1.1.50 #define SO_HDRINCL 0x02000

User access to IP header for SOCK_RAW.

2.1.1.51 #define SO_NOSLOWSTART 0x04000

Suppress slowstart on this socket.

2.1.1.52 #define SO_FULLMSS 0x08000

Not used.

2.1.1.53 #define SO_NONIP_DSSHNDL 0x10000

Set dss handle for nonip.

2.1.1.54 #define SO_NONIP_EXTYPE 0x20000

Set exception type for nonip.

2.1.1.55 #define SO_SNDBUF 0x1001

Send buffer size.

2.1.1.56 #define SO_RCVBUF 0x1002

Receive buffer size.

2.1.1.57 #define SO_SNDTIMEO 0x1005

Send a timeout.

2.1.1.58 #define SO_RCVTIMEO 0x1006

Receive a timeout.

2.1.1.59 #define SO_ERROR 0x1007

Socket error.

2.1.1.60 #define SO_RXDATA 0x1011

Get a count of bytes in sb_rcv.

2.1.1.61 #define SO_TXDATA 0x1012

Get a count of bytes in sb_snd.

2.1.1.62 #define SO_MYADDR 0x1013

Return my IP address.

2.1.1.63 #define SO_NBIO 0x1014

Set socket to Non-blocking mode.

2.1.1.64 #define SO_BIO 0x1015

Set socket to Blocking mode.

2.1.1.65 #define SO_NONBLOCK 0x1016

Set/get blocking mode via the optval parameter.

2.1.1.66 #define SO_CALLBACK 0x1017

Set/get the TCP zero_copy callback routine.

2.1.1.67 #define SO_UDPCALLBACK 0x1019

Set/get the UDP zero_copy callback routine.

2.1.1.68 #define SO_TCP_ACKDELAYTIME 0x2001

Set time for delayed acks in (ms).

2.1.1.69 #define SO_TCP_NOACKDELAY 0x2002

Suppress delayed ACKs.

2.1.1.70 #define TCP_MAXSEG 0x2003

Set maximum segment size.

2.1.1.71 #define TCP_NODELAY 0x2004

Enables TCP No Delay.

2.1.1.72 #define TCP_MAX_RXT 0x2005

Set the maximum number of rxt.

2.1.1.73 #define TCP_OT 0x2006

Set the overall timer for outstanding connection.

2.1.1.74 #define TCP_IRT 0X2007

Set the initial retxn timeout value.

2.1.1.75 #define TCP_KEEPIDLE 0X2008

Set the time before probing.

2.1.1.76 #define TCP_KEEPINTVL 0X2009

Set the probe interval.

2.1.1.77 #define TCP_KEEPCNT 0X2010

Set the max probes before drop.

2.1.1.78 #define IPPROTO_IP 0

For use with [gs]etsockopt() at IPPROTO_IP level.

2.1.1.79 #define IP_HDRINCL 2

IP header is included with the data.

2.1.1.80 #define IP_MULTICAST_IF 9

Set/get the IP multicast interface.

2.1.1.81 #define IP_MULTICAST_TTL 10

2.1.1.82 #define IP_MULTICAST_LOOP 11

Set/get the IP multicast loopback.

2.1.1.83 #define IP_ADD_MEMBERSHIP 12

Add an IPv4 group membership.

2.1.1.84 #define IP_DROP_MEMBERSHIP 13

Drop an IPv4 group membership.

2.1.1.85 #define IPV6_MULTICAST_IF 80

Set the egress interface for multicast traffic.

2.1.1.86 #define IPV6_MULTICAST_HOPS 81

Set the number of hops.

2.1.1.87 #define IPV6_MULTICAST_LOOP 82

Enable/disable loopback for multicast.

2.1.1.88 #define IPV6_JOIN_GROUP 83

Join an IPv6 MC group.

2.1.1.89 #define IPV6_LEAVE_GROUP 84

Leave an IPv6 MC group.

2.1.1.90 #define IP_EXCLUDE_LIST 17

Set/get the exclude list for 255 RAW socket.

2.1.1.91 #define IP_OPTIONS 1

For use with [gs]setsockopt() at IP_OPTIONS level.

2.1.1.92 #define IP_TOS 3

IPv4 type of service and precedence.

2.1.1.93 #define IP_TTL_OPT 4

IPv4 time to live.

2.1.1.94 #define IPV6_SCOPEID 14

IPv6 IF scope ID.

2.1.1.95 #define IPV6_UNICAST_HOPS 15

IPv6 hop limit.

2.1.1.96 #define IPV6_TCLASS 16

IPv6 traffic class.

2.1.1.97 #define MSG_OOB 0x1

Send/receive out-of-band data.

2.1.1.98 #define MSG_PEEK 0x2

Peek at the incoming message.

2.1.1.99 #define MSG_DONTROUTE 0x4

Send without using routing tables.

2.1.1.100 #define MSG_DONTWAIT 0x20

Send/receive is nonblocking.

2.1.1.101 #define MSG_ZEROCOPYSEND 0x1000

Send with zero-copy.

2.1.1.102 #define SHUT_RD 0

Shutdown "read" half.

2.1.1.103 #define SHUT_WR 1

Shutdown "write" half.

2.1.1.104 #define SHUT_RDWR 2

Shutdown both "read" and "write".

2.1.1.105 #define QAPI_NET_WAIT_FOREVER (0xFFFFFFFF)

Infinite time for the timeout_ms argument in [qapi_select\(\)](#).

2.1.1.106 #define in6_addr qapi_in6_addr

Pre-pending QAPI to internal socket structures' names to avoid conflict with internal structures.

2.1.1.107 #define FD_ZERO(set) qapi_fd_zero((set))

Clears a set.

2.1.1.108 #define FD_CLR(handle, set) qapi_fd_clr((handle), (set))

Removes a given file descriptor from a set.

2.1.1.109 #define FD_SET(handle, set) qapi_fd_set((handle), (set))

Adds a given file descriptor from a set.

2.1.1.110 #define FD_ISSET(handle, set) qapi_fd_isset((handle), (set))



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

FIBOCOM
Confidential

2.1.2 Data Structure

2.1.2.1 struct in_addr

IPv4 Internet address.

Data fields

Type	Parameter	Description
uint32_t	s_addr	IPv4 address in network order.

2.1.2.2 struct sockaddr_in

BSD-style socket IPv4 Internet address.

Data fields

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.

2.1.2.3 struct in6_addr

IPv6 internet address.

Data fields

Type	Parameter	Description
uint8_t	s_addr	128-bit IPv6 address.

2.1.2.4 struct ip46addr_n

BSD-style socket IPv6 internet address.

Data fields

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

2.1.2.5 union ip46addr_n.a

Data fields

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

2.1.2.6 union ip46addr_n.g

Data fields

Type	Parameter	Description
unsigned long	gtwy4	IPv4 gateway.
uint8_t	gtwy6	IPv6 gateway.

2.1.2.7 struct sockaddr_in6

Socket address information.

Data fields

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.

2.1.2.8 struct ip46addr

Socket IPv4/IPv6 Internet address union.

Data fields

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

2.1.2.9 union ip46addr.a

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

2.1.2.10 struct sockaddr

Generic socket internet address.

Data fields

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.

2.1.2.11 struct sockaddr_ep

Exclude list endpoint.

Data fields

Type	Parameter	Description
struct sockaddr_ep *	sockaddr_ep_next	Next endpoint.
struct sockaddr *	sockaddr_ep_addr	Endpoint address in exclude list.

2.1.2.12 struct fd_set

File descriptor sets for [qapi_select\(\)](#).

Data fields

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

2.1.2.13 struct fd_set_v2

File descriptor sets for [qapi_select\(\)](#). Increased number of file descriptors.

Data fields

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

2.1.2.14 struct linger

This option is used to set the linger time that the socket will remain in the TCP FIN WAIT 1 state before closing the socket and cleaning up the resources.

Data fields

Type	Parameter	Description
int	l_onoff	Option on/off.
int	l_linger	Linger time.

2.2 API Functions

2.2.1 qapi_socket

Creates an endpoint for communication.

Prototype

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

Parameters

in	<i>family</i>	Protocol family used for communication. The supported families are: AF_INET – IPv4 Internet protocols AF_INET6 – IPv6 Internet protocols AF_NONIP – NON IP Protocol
in	<i>type</i>	Transport mechanism used for communication. The supported types are: SOCK_STREAM – TCP SOCK_DGRAM – UDP SOCK_NONIP – NON IP
in	<i>protocol</i>	Must be set to 0.

Returns

On success, a handle for the new socket.

On error, -1.

2.2.2 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

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>addr</i>	Pointer to an address to be assigned to the socket. The actual address structure passed for the <i>addr</i> argument will depend on the address family.
in	<i>addrlen</i>	Specifies the size, in bytes, of the address pointed to by <i>addr</i> .

Returns

On success, 0.

On error, -1. Errors:

- [ENOBUFS](#): No memory available.
- [EINVAL](#): Invalid arguments passed.
- [EADDRNOTAVAIL](#): Address not available.
- [EADDRINUSE](#): Address and port are in use. [SO_REUSEADDR](#) is not set.

2.2.3 qapi_listen

Marks the socket as a passive socket.

Prototype

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

Parameters

in	<i>handle</i>	Handle (returned from qapi_socket()) that refers to a SOCK_STREAM socket.
in	<i>backlog</i>	Define the maximum length to which the queue of pending connections for the handle may grow.

Returns

On success, 0.

On error, -1. Errors:

- [ENOBUFS](#): No memory available.
- [EINVAL](#): Invalid arguments passed.
- [EADDRNOTAVAIL](#): Address not available.
- [EADDRINUSE](#): Address and port are in use. [SO_REUSEADDR](#) is not set

2.2.4 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

in	<i>handle</i>	Socket handle that has been created with qapi_socket() , bound to a local address with qapi_bind() , and listens for connections after qapi_listen() .
in	<i>cliaddr</i>	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.
in	<i>addrlen</i>	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.

Returns

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

- **EINVAL**: Invalid arguments passed to set the value.
- **EWOULDBLOCK**: If the socket is marked as non-blocking, and connection is in establishing state and not established.
- **ECONNABORTED**: Connection is aborted.

2.2.5 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 *svraddr.

Prototype

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

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>svraddr</i>	Pointer to the peer's address to which the socket is connected.
in	<i>addrlen</i>	Specify the size (in bytes) of *svraddr.

Returns

On success, 0.

On error, -1. Errors:

- **EINVAL**: Invalid arguments passed to set the value.
- **EWOULDBLOCK**: If the socket is marked as non-blocking, and connection is in establishing state and not established.
- **ECONNABORTED**: Connection is aborted.
- **EISCONN**: Connection is already established.
- **ETIMEDOUT**: Connection establishment timed out; no response from peer.
- **ENOBUFS**: No memory.

2.2.6 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

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>level</i>	Protocol level at which the option exists.
in	<i>optname</i>	Name of the option.
in	<i>optval</i>	Pointer to the option value to be set.
in	<i>optlen</i>	Option length in bytes.

Returns

On success, 0.

On error, -1. Errors:

- [EINVAL](#): Invalid arguments passed to set the value.
- [ENOTCONN](#): Tries to set socket options that require socket to be in a connected state for setting the same.
- [ENP_PARAM](#): Tries to set a socket option which is either not supported or not present.
- [ENOBUFS](#): No memory.

2.2.7 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

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>level</i>	Protocol level at which the option exists.
in	<i>optname</i>	Name of the option.
in	<i>optval</i>	Pointer to a buffer in which the value for the requested option is to be returned.
in	<i>optlen</i>	Option length in bytes.

Returns

On success, 0.

On error, -1. Errors:

- **EINVAL**: Invalid arguments passed to set the value.
- **ENOTCONN**: Tries to set socket options that require socket to be in a connected state for setting the same.
- **ENP_PARAM**: Tries to set a socket option which is either not supported or not present.
- **ENOBUFFS**: No memory

2.2.8 qapi_socketclose

Closes a socket.

Prototype

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

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
----	---------------	---

Returns

On success, 0.

On error, -1. Errors:

- [ENOTCONN](#): Socket is not connected.
- [EALREADY](#): Socket close already in progress.
- [EINVAL](#): Invalid socket parameter.

2.2.9 qapi_errno

Gets the last error code on a socket.

Prototype

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

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
----	---------------	---

Returns

Socket error code, or ENOTSOCK if socket is not found. Error:

- [EFAULT](#): Invalid socket passed to get the error.

2.2.10 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

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>buf</i>	Pointer to a buffer for the received message.
in	<i>len</i>	Number of bytes to receive.
in	<i>flags</i>	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.
in	<i>from</i>	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.
in	<i>fromlen</i>	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.

Returns

The number of bytes received, or -1 if an error occurred. Errors:

- [ENOTCONN](#): Socket is in disconnected state.
- [EWOULDBLOCK](#): If the socket is in non-blocking state, a send timeout is set, and no data is received for that duration.
- [ENOBUFS](#): No memory.

2.2.11 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

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>buf</i>	Pointer to a buffer for the received message.
in	<i>len</i>	Number of bytes to receive.
in	<i>flags</i>	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.

Returns

The number of bytes received, or -1 if an error occurred. Errors:

- [ENOTCONN](#): Socket is in disconnected state.
- [EWOULDBLOCK](#): If the socket is in non-blocking state, a send timeout is set, and no data is received for that duration.
- [EPIPE](#): For a TCP socket, shows that a peer has disconnected the socket.
- [ENOBUFS](#): No memory

2.2.12 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

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>buf</i>	Pointer to a buffer containing the message to be sent.
in	<i>len</i>	Number of bytes to send.
in	<i>flags</i>	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.
in	<i>to</i>	Pointer to the address of the target.
in	<i>tolen</i>	Size in bytes of the target address.

Returns

The number of bytes sent, or -1 if an error occurred and error number is set appropriately. Errors:

- [ENOTCONN](#): Socket is in disconnected state.
- [EWOULDBLOCK](#): If the socket is in non-blocking state, a send timeout is set, and no data is received for that duration.
- [EPIPE](#): For a TCP socket, shows that a peer has disconnected the socket.
- [ENOBUFS](#): No memory.

2.2.13 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

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

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>buf</i>	Pointer to a buffer containing message to be sent.
in	<i>len</i>	Number of bytes to send.
in	<i>flags</i>	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 – Do 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.

Returns

The number of bytes sent, or -1 if an error occurred and error number is set appropriately. Errors:

- [ENOTCONN](#): Socket is in disconnected state.
- [EWOULDBLOCK](#): If the socket is in non-blocking state, a send timeout is set, and no data is received for that duration.
- [EPIPE](#): For a TCP socket, shows that a peer has disconnected the socket.
- [ENOBUFS](#): No memory.

2.2.14 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

in	<i>rd</i>	Pointer to a list of read socket handles.
in	<i>wr</i>	Pointer to a list of write socket handles.
in	<i>ex</i>	Pointer to a list of exception socket handles.
in	<i>timeout_ms</i>	Timeout values in milliseconds.

Returns

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

2.2.15 qapi_select_v2

Enhanced qapi_select with increased number of socket file descriptors. Monitors multiple socket handles, waiting until one or more of them becomes "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 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_v2 ( fd_set_v2 * rd, fd_set_v2 * wr, fd_set_v2 * ex, int32_t timeout_ms )
```

Parameters

in	<i>rd</i>	Pointer to a list of read socket handles.
in	<i>wr</i>	Pointer to a list of write socket handles.
in	<i>ex</i>	Pointer to a list of exception socket handles.
in	<i>timeout_ms</i>	Timeout values in milliseconds.

Returns

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

2.2.16 qapi_fd_zero

Initializes a socket that is set to zero.

Prototype

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

Parameters

in	set	Pointer to a list of sockets.
----	-----	-------------------------------

Returns

On success, 0. On error, -1.

2.2.17 qapi_fd_zero_v2

Enhanced qapi_fd_zero with with increased number of socket file descriptors. Initializes a socket that is set to zero.

Prototype

```
qapi_Status_t qapi_fd_zero_v2 ( fd_set_v2 * set )
```

Parameters

in	set	Pointer to a list of sockets.
----	-----	-------------------------------

Returns

On success, 0. On error, -1.

2.2.18 qapi_fd_clr

Removes a socket from the socket set.

Prototype

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

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>set</i>	Pointer to a list of sockets.

Returns

On success, 0. On error, -1.

2.2.19 qapi_fd_clr_v2

Enhanced qapi_fd_clr with with increased number of socket file descriptors. Removes a socket from the socket set.

Prototype

```
qapi_Status_t qapi_fd_clr_v2 ( int32_t handle, fd_set_v2 * set )
```

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>set</i>	Pointer to a list of sockets.

Returns

On success, 0. On error, -1.

2.2.20 qapi_fd_set

Adds a socket to the socket set.

Prototype

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

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>set</i>	Pointer to a list of sockets.

Returns

On success, 0. On error, -1.

2.2.21 qapi_fd_set_v2

Enhanced qapi_fd_set with with increased number of socket file descriptors. Adds a socket to the socket set.

Prototype

```
qapi_Status_t qapi_fd_set_v2 ( int32_t handle, fd_set_v2 * set )
```

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>set</i>	Pointer to a list of sockets.

Returns

On success, 0. On error, -1.

2.2.22 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

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>set</i>	Pointer to a list of sockets.

Returns

On error, -1.

On success:

- 0 if the socket is not a member.
- 1 if the socket is a member.

2.2.23 qapi_fd_isset_v2

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

Prototype

```
qapi_Status_t qapi_fd_isset_v2 ( int32_t handle, fd_set_v2 * set )
```

Parameters

in	<i>handle</i>	Socket handle returned from qapi_socket() .
in	<i>set</i>	Pointer to a list of sockets.

Returns

On error, -1.

On success:

- 0 if the socket is not a member.
- 1 if the socket is a member.

2.2.24 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

in	<i>handle</i>	Socket handle returned from qapi_socket()
in	<i>addr</i>	Pointer to a user buffer of sockaraddr type which is filled by the API with the peer addr information.
in	<i>addrlen</i>	Specifies the size, in bytes, of the address pointed to by addr

Returns

On success, 0.

On error, -1. Errors:

- [ENOTCONN](#): Socket is not in a connected state.
- [EINVAL](#): Invalid arguments passed.
- [ENOBUFS](#): No memory.

2.2.25 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

in	<i>handle</i>	Socket handle returned from qapi_socket()
in	<i>addr</i>	Pointer to a user buffer of <code>sockaraddr</code> type which is filled by the API with the peer <code>addr</code> info.
in	<i>addrlen</i>	Specifies the size, in bytes, of the address pointed to by <code>addr</code>

Returns

On success, 0.

On error, -1. Errors:

- [ENOTCONN](#): Socket is not in a connected state.
- `EINVAL`: Invalid arguments passed.
- [ENOBUFS](#): No memory.

3 QAPI 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 ciphersuites.

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 contains the following sections:

- [QAPI SSL Data Types](#)
- [QAPI SSL Typedefs](#)
- [Create an SSL Object](#)
- [Create an SSL Connection Handle](#)
- [Configure an SSL Connection](#)
- [Delete an SSL Certificate](#)
- [Store an SSL Certificate](#)
- [Convert and Store an SSL Certificate](#)
- [Load an SSL Certificate](#)
- [Load an SSL Certificate and Return the Identifier](#)
- [Get a List of SSL Certificates](#)
- [Attach a Socket Descriptor to the SSL Connection](#)
- [Accept an SSL Connection From the Client](#)
- [Initiate an SSL Handshake](#)
- [Close an SSL Connection](#)
- [Free an SSL Object Handle](#)
- [Read SSL Data](#)
- [Write SSL Data](#)
- [Determine Whether a Certificate File Exists](#)
- [Set Extended Configuration Options](#)



This section provides the macros and constants, data structures, and enumerations for the networking SSL module.

3.1.1 Data Define

3.1.1.1 #define QAPI_NET_SSL_MAX_CERT_NAME_LEN (64)

Maximum number of characters in a certificate or CA list name.

3.1.1.2 #define QAPI_NET_SSL_MAX_DOMAIN_NAME_LEN (64)

Maximum number of characters in a domain name for the certificates.

3.1.1.3 #define QAPI_NET_SSL_MAX_NUM_CERTS (10)

Maximum number of file names returned in the [qapi_Net_SSL_Cert_List\(\)](#) API.

3.1.1.4 #define QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH 8

Maximum number of cipher suites that can be configured.

3.1.1.5 #define QAPI_NET_SSL_INVALID_HANDLE (0)

Invalid handle.

3.1.1.6 #define QAPI_NET_SSL_PROTOCOL_UNKNOWN 0x00

Unknown SSL protocol version.

3.1.1.7 #define QAPI_NET_SSL_PROTOCOL_TLS_1_0 0x31

TLS version 1.0.

3.1.1.8 #define QAPI_NET_SSL_PROTOCOL_TLS_1_1 0x32

TLS version 1.1.

3.1.1.9 #define QAPI_NET_SSL_PROTOCOL_TLS_1_2 0x33

TLS version 1.2.

3.1.1.10 #define QAPI_NET_SSL_PROTOCOL_TLS_1_3 0x34

TLS version 1.3.

3.1.1.11 #define QAPI_NET_SSL_PROTOCOL_DTLS_1_0 0xEF

DTLS version 1.0.

3.1.1.12 #define QAPI_NET_SSL_PROTOCOL_DTLS_1_2 0xED

DTLS version 1.2.

3.1.1.13 #define QAPI_NET_TLS_PSK_WITH_RC4_128_SHA 0x008A

TLS PSK with RC4 128 SHA.

3.1.1.14 #define QAPI_NET_TLS_PSK_WITH_3DES_EDE_CBC_SHA 0x008B

TLS PSK with 3DES EDE CBC SHA.

3.1.1.15 #define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA 0x008C

TLS PSK with AES 128 CBC SHA.

3.1.1.16 #define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA 0x008D

TLS PSK with AES 256 CBC SHA.

3.1.1.17 #define QAPI_NET_TLS_PSK_WITH_AES_128_GCM_SHA256 0x00A8

TLS PSK with AES_128 GCM SHA256.

3.1.1.18 #define QAPI_NET_TLS_PSK_WITH_AES_256_GCM_SHA384 0x00A9

TLS PSK with AES 256 GCM SHA384.

3.1.1.19 #define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA256 0x00AE

TLS PSK with AES 128 CBC SHA256.

3.1.1.20 #define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA384 0x00AF

TLS PSK with AES 256 CBC SHA384.

3.1.1.21 #define QAPI_NET_TLS_PSK_WITH_AES_128_CCM_8 0xC0A8

TLS PSK with AES 128 CCM-8.

3.1.1.22 #define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA 0x002F

Cipher TLS RSA with AES 128 CBC SHA.

3.1.1.23 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA 0x0033

Cipher TLS DHE RSA with AES 128 CBC SHA.

3.1.1.24 #define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA 0x0035

Cipher TLS RSA with AES 256 CBC SHA.

3.1.1.25 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA 0x0039

Cipher TLS DHE RSA with AES 256 CBC SHA.

3.1.1.26 #define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA256 0x003C

Cipher TLS RSA with AES 128 CBC SHA256.

3.1.1.27 #define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA256 0x003D

Cipher TLS RSA with AES 256 CBC SHA256.

3.1.1.28 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 0x0067

Cipher TLS DHE RSA with AES 128 CBC SHA256.

3.1.1.29 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 0x006B

Cipher TLS DHE RSA with AES 256 CBC SHA256.

3.1.1.30 #define QAPI_NET_TLS_RSA_WITH_AES_128_GCM_SHA256 0x009C

Cipher TLS RSA with AES 128 GCM SHA256.

3.1.1.31 #define QAPI_NET_TLS_RSA_WITH_AES_256_GCM_SHA384 0x009D

Cipher TLS RSA with AES 256 GCM SHA384.

3.1.1.32 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 0x009E

Cipher TLS DHE RSA with AES 128 GCM SHA256.

3.1.1.33 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 0x009F

Cipher TLS DHE RSA with AES 256 GCM SHA384.

3.1.1.34 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA 0xC004

Cipher TLS ECDH ECDSA with AES 128 CBC SHA.

3.1.1.35 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA 0xC005

Cipher TLS ECDH ECDSA with AES 256 CBC SHA.

**3.1.1.36 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
0xC009**



Cipher TLS ECDHE ECDSA with AES 128 CBC SHA.

**3.1.1.37 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
0xC00A**

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA.

3.1.1.38 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA 0xC00E

Cipher TLS ECDH RSA with AES 128 CBC SHA.

3.1.1.39 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA 0xC00F

Cipher TLS ECDH RSA with AES 256 CBC SHA.

3.1.1.40 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA 0xC013

Cipher TLS ECDHE RSA with AES 128 CBC SHA.

3.1.1.41 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA 0xC014

Cipher TLS ECDHE RSA with AES 256 CBC SHA.

**3.1.1.42 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 0x-
C023**

Cipher TLS ECDHE ECDSA with AES 128 CBC SHA256.

**3.1.1.43 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 0x-
C024**

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA384.

3.1.1.44 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 0xC025

Cipher TLS ECDH ECDSA with AES 128 CBC SHA256.

3.1.1.45 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 0xC026

Cipher TLS ECDH ECDSA with AES 256 CBC SHA384.

3.1.1.46 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 0xC027

Cipher TLS ECDHE RSA with AES 128 CBC SHA256.

3.1.1.47 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 0xC028

Cipher TLS ECDHE RSA with AES 256 CBC SHA384.

3.1.1.48 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 0xC029

Cipher TLS ECDH RSA with AES 128 CBC SHA256.

3.1.1.49 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 0xC02A

Cipher TLS ECDH RSA with AES 256 CBC SHA384.

3.1.1.50 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 0xC02B

Cipher TLS ECDHE ECDSA with AES 128 GCM SHA256.

3.1.1.51 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 0xC02C

Cipher TLS ECDHE ECDSA with AES 256 GCM SHA384.

3.1.1.52 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 0xC02D

Cipher TLS ECDH ECDSA with AES 128 GCM SHA256.

3.1.1.53 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 0xC02E

Cipher TLS ECDH ECDSA with AES 256 GCM SHA384.

3.1.1.54 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 0xC02F

Cipher TLS ECDHE RSA with AES 128 GCM SHA256.

3.1.1.55 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 0xC030

Cipher TLS ECDHE RSA with AES 256 GCM SHA384.



3.1.1.56 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 0xC031

Cipher TLS ECDH RSA with AES 128 GCM SHA256.

3.1.1.57 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 0xC032

Cipher TLS ECDH RSA with AES 256 GCM SHA384.

3.1.1.58 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 0xC0AE

Cipher TLS ECDHE ECDSA With AES 128 CCM8.

3.1.1.59 #define QAPI_NET_TLS_RSA_WITH_AES_128_CCM 0xC09C

Cipher TLS RSA with AES 128 CCM.

3.1.1.60 #define QAPI_NET_TLS_RSA_WITH_AES_256_CCM 0xC09D

Cipher TLS RSA with AES 256 CCM.

3.1.1.61 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM 0xC09E

Cipher TLS DHE RSA with AES 128 CCM.

3.1.1.62 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM 0xC09F

Cipher TLS DHE RSA with AES 256 CCM.

3.1.1.63 #define QAPI_NET_TLS_RSA_WITH_AES_128_CCM_8 0xC0A0

Cipher TLS RSA with AES 128 CCM 8.

3.1.1.64 #define QAPI_NET_TLS_RSA_WITH_AES_256_CCM_8 0xC0A1

Cipher TLS RSA with AES 256 CCM 8.

3.1.1.65 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM_8 0xC0A2

Cipher TLS DHE RSA with AES 128 CCM 8.

3.1.1.66 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM_8 0xC0A3

Cipher TLS DHE RSA with AES 256 CCM 8.

3.1.1.67 #define QAPI_NET_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC13

Cipher TLS ECDHE RSA with CHACHA20 POLY1305 SHA256.

3.1.1.68 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 0xCC14

Cipher TLS ECDHE ECDSA with CHACHA20 POLY1305 SHA256.

3.1.1.69 #define QAPI_NET_TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC15

Cipher TLS DHE RSA with CHACHA20 POLY1305 SHA256. TLS1.3 Cipher AES 128 GCM SHA256.

3.1.1.70 #define QAPI_NET_TLS13_AES_128_GCM_SHA256 0x1301

TLS1.3 Cipher AES 256 GCM SHA384.

3.1.1.71 #define QAPI_NET_TLS13_AES_256_GCM_SHA384 0x1302

TLS1.3 Cipher CHACHA20 POLY1305 SHA256.

3.1.1.72 #define QAPI_NET_TLS13_CHACHA20_POLY1305_SHA256 0x1303

TLS1.3 Key Exchange group secp256r1.

3.1.1.73 #define QAPI_NET_TLS13_KEY_EXCHANGE_GROUP_SECP256R1 23

TLS1.3 Key Exchange group secp384r1.

3.1.1.74 #define QAPI_NET_TLS13_KEY_EXCHANGE_GROUP_SECP384R1 24

TLS1.3 Key Exchange group secp512r1.

3.1.1.75 #define QAPI_NET_TLS13_KEY_EXCHANGE_GROUP_SECP521R1 25

TLS1.3 Key Exchange group x25519.

3.1.1.76 #define QAPI_NET_SSL_MAX_CA_LIST 10

Maximum certificate authority list entries allowed for conversion to binary format.

3.1.2 Data Structure

3.1.2.1 struct qapi_Net_SSL_Verify_Policy_t

Structure to specify the certificate verification policy.

Data fields

Type	Parameter	Description
uint8_t	domain	TRUE to verify certificate commonName against the peer's domain name.
uint8_t	time_Verify	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 .

3.1.2.2 struct qapi_Net_SSL_Identifier_t

Structure to get the identifier from the certificate.

Data fields

Type	Parameter	Description
qapi_Net_SSL_Identifier_Type_t	identifier_Type	Type of identifier to extract from the certificate.
char	identifier_Name	Name (altDNSNames, altURIs, or common name of the certificate. See QAPI_NET_SSL_MAX_CERT_NAME_LEN .

3.1.2.3 struct qapi_Net_SSL_Config_t

Structure to configure an SSL connection.

Data fields

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.

3.1.2.4 struct qapi_Net_SSL_Cert_List_t

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

Data fields

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

3.1.2.5 struct qapi_Net_SSL_CERT_t

SSL client certificate info for conversion and storage.

Data fields

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.

3.1.2.6 struct qapi_NET_SSL_CA_Info_t

SSL certificate authority list information.

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

3.1.2.7 struct qapi_Net_SSL_CA_List_t

SSL certificate authority information for conversion and storage.

Data fields

Type	Parameter	Description
uint32_t	ca_Cnt	Certificate authority list count.
qapi_NET_SS-L_CA_Info_t *	ca_Info	Certificate authority list info.

3.1.2.8 struct qapi_Net_SSL_PSK_Table_t

SSL PSK table information for conversion and storage.

Data fields

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

3.1.2.9 struct qapi_Net_SSL_DI_Cert_t

SSL domain-issued certificate information for conversion and storage.

Data fields

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

3.1.2.10 struct qapi_Net_SSL_RPK_Cert_t

SSL client raw public key certificate information for conversion and storage.

Data fields

Type	Parameter	Description
uint8_t *	pubkey_Buf	Public key buffer.
uint32_t	pubkey_Size	Public key buffer size.
uint8_t *	privkey_Buf	Private key buffer.
uint32_t	privkey_Size	Private key buffer size.
uint8_t *	pass_Key	Password phrase.

3.1.2.11 struct qapi_Net_SSL_Cert_Info_s

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

Data fields

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

3.1.2.12 union qapi_Net_SSL_Cert_Info_s.info

Data fields

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.

Type	Parameter	Description
------	-----------	-------------

qapi_Net_SSL-DI_Cert_t	di_cert	Domain-issued certificate.
qapi_Net_SSL-RPK_Cert_t	rpk_cert	Raw public key certificate.

FIBOCOM
Confidential

3.1.3 Data Typedef

3.1.3.1 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\(\)](#).

3.1.3.2 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\(\)](#).

3.1.3.3 typedef const void*qapi_Net_SSL_Cert_t

Internal certificate format. The certificate is in a binary format optimized for speed and size. The .bin format certificate can be created using the command line tool [SharkSslParseCert].

Usage

SharkSslParseCert <cert file> <privkey file> [-p <passkey>] [-b <binary output file>]

3.1.3.4 typedef const void*qapi_Net_SSL_DICERT_t

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

3.1.3.5 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. The list can be created using the command line tool [SharkSSLParseCAList].

Usage



SharkSSLParseCAList [-b <binary output file>] <certfile> [certfile...] where certfile is a .PEM file containing one or more certificates.

3.1.3.6 typedef const void*qapi_Net_SSL_PSKTable_t

Internal psk_table format. PSK table is in an optimized binary format. The table can be created by using the command line tool [SharkSslParsePSKTable]. Set the PSK file format before using the tool.

Identity_1: psk_key1 Identity_2: psk_key2

Usage

SharkSslParsePSKTable <PSK file> [-b <binary output file>]

3.1.4 Data Enumeration

3.1.4.1 enum qapi_Net_SSL_Role_t

SSL object role.

Enumerator:

QAPI_NET_SSL_SERVER_E	Server role. Not supported.
QAPI_NET_SSL_CLIENT_E	Client role.

3.1.4.2 enum qapi_Net_SSL_Protocol_t

SSL protocol.

Enumerator:

QAPI_NET_SSL_TLS_E	TLS protocol.
QAPI_NET_SSL_DTLS_E	DTLS protocol.

3.1.4.3 enum qapi_Net_SSL_Cert_Type_t

SSL certificate type.

Enumerator:

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 certificate type
QAPI_NET_SSL_RPK_CERT_E	Raw public key certificate type

3.1.4.4 enum qapi_Net_SSL_Extended_Config_Options_t



Extended configurations items for SSL connection.

Enumerator:

QAPI_NET_SSL_EXTENDED_CONFIG_SESSION_NAME Session Name prefix used to store TLS/DTLS session.

QAPI_NET_SSL_EXTENDED_CONFIG_SESSION_LIFETIME Session Lifetime.

Data type is uint32(seconds), value range [0, 86400], default is 86400 (24 hours).

QAPI_NET_SSL_EXTENDED_CONFIG_CONNECTION_TIMEOUT Connection timeout.

Data type is uint32 (seconds), value range [1, 60], default is 60. For TLS, the value is the connection time out value.

For DTLS, the value is the max-retransmit-timer(retransmit logic: 1s, 2s, 4s,...max-retansmit-timer).

QAPI_NET_SSL_EXTENDED_CONFIG_DISABLE_CLOSE_NOTIFY Disable Close-Notify when shutdown TLS/DTLS connection. Data type is uint32, value range [0, 1].

1: disable close-notify

0: enable close-notify (default value).

QAPI_NET_SSL_EXTENDED_CONFIG_MIN_TLS_PROTOCOL set minimum TLS protocol to support Data type is uint8, value range as TLS protocol version value.

QAPI_NET_SSL_EXTENDED_CONFIG_TLS13_KEY_EXCHANGE_GROUP set key-exchange algorithm for TLS13 Data type is uint8[], each item is the enum value of key-exchange group(QAPI_NET_TLS13_KEY_EXCHANGE_GROUP_XXX).

QAPI_NET_SSL_EXTENDED_CONFIG_TLS13_KEY_EXCHANGE_NEGOTIATE enable/disable KeyExchange negotiate in TLS13 Data type is uint8, value range [0, 1].

1: Enables negotiate.

0: Disables negotiate.

QAPI_NET_SSL_EXTENDED_CONFIG_ALLOW_SERVER_RPK Whether allow server to pick raw public key certificate.for server authentication Data type is uint32, value range [0, 1].

1: Allows server to use raw public key.

0: Does not allow server to use raw public key.

3.1 API Functions

3.1.1 qapi_Net_SSL_Obj_New

Creates a new SSL object (client).

Prototype

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

Parameters

in	<i>role</i>	Client role. Server is not supported.
----	-------------	---------------------------------------

Returns

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

3.1.2 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

in	<i>hdl</i>	SSL object handle.
in	<i>prot</i>	Protocol to be used for this connection.

Returns

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

3.1.3 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 set max_frag_len is 0, the default configuration will be applied. Default configuration is:

- max_frag_len = 1024
- max_frag_len_neg_disable = 0

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 can choose to configure max_frag_len and set 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. When value 16 KB, the max_frag_len is set to 16384 and 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

in	<i>ssl</i>	Connection handle.
in	<i>cfg</i>	Configuration parameters.

Returns

QAPI_OK on success.

QAPI_ERR_INVALID_PARAM_SSL if an error occurred (configuration is invalid).

3.1.4 qapi_Net_SSL_Cert_delete

Deletes an encrypted certificate or 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

in	<i>name</i>	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.
in	<i>type</i>	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.

Returns

0 on success.

Negative value on error (see SslErrors).

3.1.5 qapi_Net_SSL_Cert_Store

Stores an internal certificate or CA list, or a PSK table in nonvolatile memory in encrypted form.

The certificate is in binary format optimized for speed and size. The .bin format certificate can be created using the command line tool [SharkSslParseCert].

The CA list is in binary format optimized for speed and size. The list can be created using the command line tool [SharkSSLParseCAList].

The PSK table is in an optimized binary format. The table can be created using the command line tool [SharkSslParsePSKTable]. Set the table format before using the tool:

Identity_1: psk_key1 Identity_2: psk_key2

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

in	<i>name</i>	Name of the certificate, CA list, or PSK table. The maximum length of the name allowed is QAPI_NET_SSL_MAX_CERT_NAME_LEN, including the NULL character.
in	<i>type</i>	Type of data (certificate, CA list, or PSK table) to store. Could be either QAPI_NET_SSL_CERTIFICATE_E, QAPI_NET_SSL_CA_LIST_E, or QAPI_NET_SSL_PSK_TABLE_E.
in	<i>cert</i>	Address of the file containing the certificate in SSL internal format (*.bin file).
in	<i>size</i>	Size of the certificate file.

Returns

0 on success.

Negative value on error (see SslErrors).

3.1.6 qapi_Net_SSL_Cert_Convert_And_Store

Converts certificates, CA lists from .PEM 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.

The PSK table must be in the following format:

- Identity_1: psk_key1
- Identity_2: psk_key2

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

in	<i>cert_info</i>	Information pertaining to either the client certificate, CA lists in .PEM format or PSK tables.
in	<i>cert_name</i>	Name of the certificate, CA list, or PSK table that the cert_info is to be stored under after the conversion.

Returns

0 on success.

Negative value on error (see SslErrors).

3.1.7 qapi_Net_SSL_Cert_Load

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

- Certificate – Loads a client certificate to the SSL object.
- 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 conext. 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

in	<i>hdl</i>	SSL object handle.
in	<i>type</i>	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.
in	<i>name</i>	Name of the file to load.

Returns

0 on success.

Negative value on error (see SslErrors)

3.1.8 qapi_Net_SSL_Cert_Load_Get_Identifier

Reads an encrypted domain-issued certificate (RFC6698, mode 3) from nonvolatile memory, decrypts it, and then adds it to the SSL object and returns the identifier.

Domain Issued Certificate: Load a DI certificate to the SSL object.

Prototype

```
qapi_Status_t qapi_Net_SSL_Cert_Load_Get_Identifier ( qapi_Net_SSL_Obj_Hdl_t hdl,
qapi_Net_SSL_Identifier_t * identifier, const char * name)
```

Parameters

in	<i>hdl</i>	SSL object handle.
out	<i>identifier</i>	Type of certificate identifier.
in	<i>name</i>	Name of the file to load.

Returns

0 on success,

Negative value on error (see SslErrors).

3.1.9 qapi_Net_SSL_Cert_List

Gets a list of encrypted certificates or CA lists, or a PSK table stored in nonvolatile memory. The structure [qapi_Net_SSL_Cert_List_t](#) 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

in	<i>type</i>	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.
in,out	<i>list</i>	List of file names.

Returns

Number of files. 0 on error.

3.1.10 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

in	<i>ssl</i>	SSL connection handle.
in	<i>fd</i>	FD socket descriptor.

Returns

QAPI_OK on success. QAPI_ERR_INVALID_PARAM_SSL on error.

3.1.11 qapi_Net_SSL_Accept

Deprecated API is deprecated in dataservices version 1.6.0. 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

in	ssl/	SSL connection handle.
----	------	------------------------

Returns

QAPI_SSL_OK_HS on success.

QAPI_ERR_* on error.

3.1.12 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

in	ssl/	SSL connection handle.
----	------	------------------------

Returns

QAPI_SSL_OK_HS on success.

QAPI_ERR_* on error

3.1.13 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 socket QAPIs.

Prototype

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

Parameters

in	ssl/	SSL connection handle.
----	------	------------------------

Returns

QAPI_OK on success.

QAPI_ERR_INVALID_PARAM_SSL on error (invalid connection handle).

3.1.14 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

in	<i>hdl</i>	SSL object handle.
----	------------	--------------------

Returns

QAPI_OK on success.

Dependencies

All connections belonging to this handle must be closed before calling this API.

3.1.15 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

in	<i>hdl</i>	Connection handle.
in,out	<i>buf</i>	Buffer to hold received data. Must be allocated by the application.
in	<i>size</i>	Size of the buffer in bytes.

Returns

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.

3.1.16 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

in	<i>hdl</i>	Connection handle.
in	<i>buf</i>	Buffer with the data to be sent.
in	<i>size</i>	Size of buf in bytes.

Returns

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.

3.1.17 qapi_Net_SSL_Cert_File_Exists

Given the certificate name and type, returns whether the file exists in the encrypted location.

Prototype

```
qapi_Status_t qapi_Net_SSL_Cert_File_Exists ( char * file_name,qapi_Net_SSL_Cert_Type_t type )
```

Parameters

in	<i>file_name</i>	Certificate file name.
in	<i>type</i>	Type of SSL certificate.

Returns

QAPI_OK on success.

QAPI_ERR_* on error.

3.1.18 qapi_Net_SSL_Set_Extended_Config_Option

Set extended configuration options for SSL connection.

Prototype

```
qapi_Status_t qapi_Net_SSL_Set_Extended_Config_Option ( qapi_Net_SSL_Con_Hdl_t hdl,
qapi_Net_SSL_Extended_Config_Options_t option, void *val, uint32_t len )
```

Parameters

in	<i>hdl</i>	Connection handle.
in	<i>option</i>	Extended option enum.
in	<i>val</i>	Variable pointer for the option.
in	<i>len</i>	Length of val.

Returns

QAPI_OK on success.

QAPI_ERR_* on error.

4 QAPI Networking Services

This chapter describes the Networking Services and utilities QAPIs.

- [Networking Data Types](#)
- [Get the Names of All Network Interfaces](#)
- [Parse an Address String into an IPv4/IPv6 Address](#)
- [Format an IPv4/IPv6 Address into a NULL-terminated String](#)
- [Get the Physical Address and Length of an Interface](#)
- [Check Whether an Interface Exists](#)
- [IPv4 Network Configuration](#)
- [Send an IPv4 Ping](#)
- [Send an IPv4 Ping with a Response](#)
- [IPv4 Route Commands](#)
- [Send an IPv6 Ping](#)
- [Send an IPv6 Ping with a Response](#)
- [Get the IPv6 Address of an Interface](#)
- [IPv6 Route Commands](#)
- [Get the Interface Scope ID](#)

4.1 Networking Data Types

This section provides the macros and constant, data structures, and enumerations for the networking services module.

4.1.1 Data Define

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

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.

Parameters

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

Returns

1 if the IPv4 address is multicast, 0 otherwise.

4.1.1.2 #define IF_NAMELEN 20

Default maximum length for interface names.

4.1.1.3 #define QAPI_NET_IPV4_MAX_ROUTES (3)

Maximum IPv4 routing configurations.

4.1.1.4 #define QAPI_IS_IPV6_LINK_LOCAL(ipv6_Address)

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 111111010, followed by 54 zeros, followed by 64 bits of the interface identifier.

Parameters

in	<i>ipv6_Address</i>	IPv6 address to check.
----	---------------------	------------------------

Returns

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

4.1.1.5 #define QAPI_IS_IPV6_MULTICAST(*ipv6_Address*) (*ipv6_Address*[0] == 0xff)

Checks whether the IPv6 address is multicast.

Parameters

in	<i>ipv6_Address</i>	IPv6 address to check.
----	---------------------	------------------------

Returns

1 if the IPv6 address is multicast, 0 otherwise.

4.1.1.6 #define QAPI_NET_IPV6_MAX_ROUTES (3)

Maximum IPv6 routing configurations.

4.1.1.7 #define QAPI_NET_IFNAME_LEN 12

Maximum length for the interface name.

4.1.1.8 #define QAPI_NET_ICMP_ECHOREP 0

ICMP type values. ICMP Echo reply.

4.1.1.9 #define QAPI_NET_ICMP_DESTIN 3

Destination Unreachable.

4.1.1.10 #define QAPI_NET_ICMP_SOURCEQ 4

Source quench.

4.1.1.11 #define QAPI_NET_ICMP_REDIR 5

Redirect.

4.1.1.12 #define QAPI_NET_ICMP_ECHOREQ 8

ICMP Echo request.

4.1.1.13 #define QAPI_NET_ICMP_TIMEX 11

Time exceeded.

4.1.1.14 #define QAPI_NET_ICMP_PARAM 12

Parameter problem.

4.1.1.15 #define QAPI_NET_ICMP_TIMEREQ 13

Timestamp request.

4.1.1.16 #define QAPI_NET_ICMP_TIMEREP 14

Timestamp reply.

4.1.1.17 #define QAPI_NET_ICMP_INFO 15

Information request.

4.1.1.18 #define QAPI_NET_ICMPV6_ECHOREP 129

ICMPV6 Echo reply.

4.1.1.19 #define QAPI_NET_ICMP_DESTIN_NETWORK_UNREACHABLE 0

Destination unreachable error code. Destination network unreachable.

4.1.1.20 #define QAPI_NET_ICMP_DESTIN_HOST_UNREACHABLE 1

Destination host unreachable.

4.1.1.21 #define QAPI_NET_ICMP_DESTIN_PROTOCOL_UNREACHABLE 2

Destination protocol unreachable.

4.1.1.22 #define QAPI_NET_ICMP_DESTIN_PORT_UNREACHABLE 3

Destination port unreachable.

4.1.1.23 #define QAPI_NET_ICMP_DESTIN_FRAG_REQUIRED 4

Fragmentation required, and DF flag set.

4.1.1.24 #define QAPI_NET_ICMP_DESTIN_SOURCE_ROUTE_FAILED 5

Source route failed.

4.1.1.25 #define QAPI_NET_ICMP_DESTIN_NETWORK_UNKNOWN 6

Destination network unknown.

4.1.1.26 #define QAPI_NET_ICMP_DESTIN_HOST_UNKNOWN 7

Destination host unknown.

4.1.1.27 #define QAPI_NET_ICMP_DESTIN_SOURCE_HOST_ISOLATED 8

Source host isolated.

4.1.1.28 #define QAPI_NET_ICMP_DESTIN_NETWORK_ADMIN_PROHIBIT 9

Network administratively prohibited.

4.1.1.29 #define QAPI_NET_ICMP_DESTIN_HOST_ADMIN_PROHIBIT 10

Host administratively prohibited.

4.1.1.30 #define QAPI_NET_ICMP_DESTIN_NETWORK_UNREACHABLE_TOS 11

Network unreachable for ToS.

4.1.1.31 #define QAPI_NET_ICMP_DESTIN_HOST_UNREACHABLE_TOS 12

Host unreachable for ToS.

4.1.1.32 #define QAPI_NET_ICMP_DESTIN_COMM_ADMIN_PROHIBIT 13

Communication administratively prohibited.

4.1.1.33 #define QAPI_NET_ICMP_DESTIN_HOST_PRECEDENCE_VIOLATION 14

Host precedence violation.

4.1.1.34 #define QAPI_NET_ICMP_DESTIN_PRECEDENCE_CUTOFF_IN_PROG 15

Precedence cutoff in effect.

4.1.1.35 #define QAPI_NET_ICMP_REDIR_DATAGRAM_NETWORK 0

Redirect error code. Redirect datagram for the Network.

4.1.1.36 #define QAPI_NET_ICMP_REDIR_DATAGRAM_HOST 1

Redirect datagram for the Host.

4.1.1.37 #define QAPI_NET_ICMP_REDIR_DATAGRAM_ToS_NETWORK 2

Redirect datagram for the ToS & network.

4.1.1.38 #define QAPI_NET_ICMP_REDIR_DATAGRAM_ToS_HOST 3

Redirect datagram for the ToS & host.

4.1.1.39 #define QAPI_NET_ICMP_TIMEX_TTL_EXPIRED 0

Time exceeded error code. TTL expired in transit.

4.1.1.40 #define QAPI_NET_ICMP_TIMEX_FRAG_ASSEMBLY_TIME_EXCEEDED 1

Fragment reassembly time exceeded.

4.1.1.41 #define QAPI_NET_ICMP_PARAM_POINTER_ERROR 0

4.1.1.42 #define QAPI_NET_ICMP_PARAM_REQUIRE_PARAM_MISSING 1

Missing a required option.

4.1.1.43 #define QAPI_NET_ICMP_PARAM_BAD_LENGTH 2

Bad length.

4.1.2 Data Structure

4.1.2.1 struct qapi_Net_Ping_V4_t

IPv4 ping input.

Data fields

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).

4.1.2.2 struct qapi_Net_Ping_V4_R2_t

IPv4 ping input.

Data fields

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.

4.1.2.3 struct qapi_Net_IPv4_Route_t

IPv4 routing object.

Data fields

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	ipRouteNext-Hop	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.

4.1.2.4 struct qapi_Net_IPv4_Route_List_t

IPv4 routing objects list.

Data fields

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.

4.1.2.5 struct qapi_Net_Ping_V6_s

IPv6 ping input.

Data fields

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.

FIBOCOM
Confidential

4.1.2.6 struct qapi_Net_Ping_V6_R2_t

IPv4 ping input.

Data fields

Type	Parameter	Description
uint32_t	bitmask	Bitmask.
uint32_t	ipv6_addr	Destination to ping.
uint32_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.

4.1.2.7 struct qapi_Net_IPv6_Route_t

IPv6 routing object.

Data fields

Type	Parameter	Description
uint8_t	ipv6RouteDest	Destination IPv6 address of this route.
uint32_t	ipv6RoutePfx- Length	Indicates the prefix length of the destination address.
uint8_t	ipv6RouteNext- Hop	Address of the next system en route.
uint32_t	ipv6Route- Protocol	Routing mechanism via which this route was learned.
uint32_t	ipv6Routelf-	Index value that uniquely identifies the local interface through

	Index	which the next hop of this route should be reached.
char	ifName	Textual name of the interface.

FIBOCOM
Confidential

4.1.2.8 struct qapi_Net_IPv6_Route_List_t

IPv6 routing objects list.

Data fields

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 .

4.1.2.9 struct qapi_Net_Ifnameindex_t

Network interface object.

Data fields

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).

4.1.2.10 struct qapi_Ping_Info_Resp_s

Ping response structure.

Data fields

Type	Parameter	Description
------	-----------	-------------

int	ptype	ICMP type for the ping.
int	pcode	ICMP code for the ping.
char	perror	Response description for the ping.

4.1.3 Data Enumeration

4.1.3.1 enum qapi_Net_Route_Command_t

Commands for routing QAPI net services.

Enumerator:

QAPI_NET_ROUTE_ADD_E Add route. **QAPI_NET_ROUTE_DEL_E** Delete route.
QAPI_NET_ROUTE_SHOW_E Show routes.

4.1.3.2 enum qapi_Net_IPv4cfg_Command_t

Commands for the IPv4 configuration QAPI.

Enumerator:

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).

4.1.3.3 struct qapi_Ping_Info_Resp_R2_s

Ping response structure.

Data fields

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.

4.2 API Functions

4.2.1 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

out	<i>if_Name_Index</i>	Array to contain the retrieved information.
-----	----------------------	---

Returns

Number of network interface

4.2.2 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

in	<i>af</i>	Address family. AF_INET for IPv4, AF_INET6 for IPv6.
in	<i>src</i>	IPv4 or IPv6 address string (NULL terminated).
out	<i>dst</i>	Resulting IPv4/IPv6 address.

Returns

0 if OK, 1 if bad address format, -1 if af is not AF_INET or AF_INET6.

4.2.3 const char* 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

in	<i>af</i>	Address family; AF_INET for IPv4, AF_INET6 for IPv6.
in	<i>src</i>	Pointer to an IPv4 or IPv6 address.
out	<i>dst</i>	Pointer to the output buffer to contain the IPv4/IPv6 address string.
out	<i>size</i>	Size of the output buffer in bytes.

Returns

Pointer to the resulting string if OK, otherwise NULL.

4.2.4 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. This function does not allocate space for the address, and therefore the caller must not free it.

```
int status;
const char * address = 0; uint32_t address_length = 0;
status = qapi_Net_Interface_Get_Physical_Address(interface_name, &address
, &address_length); if ( status == 0 ) {
// at this point address contains the physical address and
// address_length contains the physical address length
// address[0] is the MSB of the physical address
}
```

Prototype

```
int32_t qapi_Net_Interface_Get_Physical_Address ( const char * interface_ - Name, const uint8_t **
address, uint32_t * address_Len )
```

Parameters

in	<i>interface_Name</i>	Name of the interface for which to retrieve the physical address and or physical address length.
out	<i>address</i>	Pointer to where to save the address of the buffer containing the physical address.
out	<i>address_Len</i>	Pointer to where to store the physical address length.

Returns

0 on success, or a negative error code on failure.

4.2.5 qapi_Net_Interface_Exist

Checks whether the interface exists.

```
int exist;
```

```
exist = qapi_Net_Interface_Exist("rmnet_data0");  
if ( exist == 1 )  
{  
    printf("rmnet_data0 exists\r\n");  
}
```

Prototype

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

Parameters

in	<i>interface_Name</i>	Name of the interface for which to check whether it exists.
----	-----------------------	---

Returns

0 if the interface does not exist or 1 if the interface does exist.

4.2.6 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

in	<i>interface_Name</i>	Pointer to the interface name.
in	<i>cmd</i>	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.
in	<i>ipv4_Addr</i>	Pointer to the IPv4 address in host order.
in	<i>subnet_Mask</i>	Pointer to the IPv4 subnet mask in host order.
in	<i>gateway</i>	Pointer to the IPv4 gateway address in host order.

Returns

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

4.2.7 qapi_Net_Ping

Sends an IPv4 ping.

Prototype

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

Parameters

in	<i>ipv4_Addr</i>	IPv4 destination address in network order.
in	<i>size</i>	Size of the ping payload in bytes.

Returns

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

4.2.8 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

in	<i>ping_buf</i>	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).
out	<i>ping_resp</i>	Pointer to where to store the ping response code and the type for the ICMP echo response received.

Returns

QAPI_OK – Successful ping response is received.

QAPI_ERROR – The response buffer is filled with an error code.

4.2.9 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

in	<i>interface_Name</i>	Pointer to the interface name.
in	<i>cmd</i>	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.
in	<i>ipv4_Addr</i>	Pointer to the IPv4 address in host order.
in	<i>subnet_Mask</i>	Pointer to the IPv4 subnet mask in host order.
in	<i>gateway</i>	Pointer to the IPv4 gateway address in host order.
in	<i>route_List</i>	Pointer to the buffer to contain the list of routes, returned with the QAPI_NET_ROUTE_SHOW_E command.

Returns

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

4.2.10qapi_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

in	<i>ipv6_Addr</i>	IPv6 address to which to send a ping.
in	<i>size</i>	Number of data bytes to send.
in	<i>interface_Name</i>	Pointer to the interface name; the interface name is required when pinging an IPv6 link local address.

Returns

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

4.2.11 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

in	<i>ping6_buf</i>	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.
out	<i>ping_resp</i>	Pointer to where to store the ping response code and the type for the ICMP echo response received.

Returns

QAPI_OK – A successful ping response is received.

QAPI_ERROR – The error and response buffer is filled with the error code.

4.2.12qapi_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

in	<i>interface_Name</i>	Pointer to the name of the network interface.
in	<i>link_Local</i>	Pointer to the first global unicast address.
in	<i>global</i>	Pointer to the link local unicast address.
in	<i>default_Gateway</i>	Pointer to the default gateway address.
in	<i>global_Second</i>	Pointer to the second global unicast address.
in	<i>link_Local_Prefix</i>	Pointer to the prefix length of the link-local address.
in	<i>global_Prefix</i>	Pointer to the prefix length of the first global address.
in	<i>default_Gateway_ - Prefix</i>	Pointer to the prefix length of the default gateway address.
in	<i>global_Second_ - Prefix</i>	Pointer to the prefix length of the second global address.

Returns

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

4.2.13 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

in	<i>interface_Name</i>	Pointer to the name of the network interface.
in	<i>cmd</i>	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
in	<i>ipv6_Addr</i>	Pointer to the IPv6 address.
in	<i>prefix_Length</i>	Pointer to the IPv6 prefix length.
in	<i>next_Hop</i>	Pointer to the IPv6 gateway address.
in	<i>route_List</i>	Pointer to the buffer containing a list of routes, returned with the QAPI_NET_ROUTE_SHOW_E command.

Returns

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

4.2.14 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

in	<i>interface_Name</i>	Pointer to the name of the interface for which to retrieve the scope ID.
out	<i>scope_ID</i>	Pointer to the location store the scope ID.

Returns

0 on success, or a negative error code.

5 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 describes the following APIs:

- [DNS Client Service Macros, Data Types, and Enumerations](#)
- [Check Whether the DNS Client has Started](#)
- [Start, Stop, or Disable the DNS Client](#)
- [Convert an IP Address Text String into an IP Address](#)
- [Convert an IP Address Text String for an Interface](#)
- [Get a List of DNS Servers](#)
- [Get Index for Added DNS Server](#)
- [Add a DNS Server](#)
- [Add a DNS Server to an Interface](#)
- [Remove a DNS Server](#)
- [Removes a DNS Server from an Interface](#)
- [Get IPv4 Host Information by Name](#)
- [Get IPv4/IPv6 Host Information by Name](#)

5.1 DNS Data Types

This section provides the macros and constant, data structures, and enumerations for the DNS client service module.

5.1.1 Data Define

5.1.1.1 #define TXM_QAPI_DSS_BASE TXM_QAPI_DATA_SERVICES_BASE

Maximum IDs for dataservices is defined by macro TXM_QAPI_DATA_SERVICES_NUM_IDS.

5.1.1.2 #define MAX_DNS_SVR_NUM 4

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

5.1.1.3 #define QAPI_MAX_DNS_ADDRS 5

Maximum number of DNS addresses in h_addr_list of qapi_hostent_s.

5.1.1.4 #define QAPI_MAX_DNS_NAME 256

Maximum length of DNS hostname.

5.1.1.5 #define QAPI_MAX_IPADDR_LEN 16

Maximum length of individual h_addr_list ipaddress string of qapi_hostent_s.

5.1.1.6 #define QAPI_DNS_PORT 53

5.1.1.7 #define QAPI_NET_DNS_ANY_SERVER_ID 0xFFFF

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.

5.1.1.8 #define QAPI_NET_DNS_V4_PRIMARY_SERVER_ID 0

DNS IPv4 primary server ID.

5.1.1.9 #define QAPI_NET_DNS_V4_SECONDARY_SERVER_ID 1

DNS IPv4 secondary server ID.

5.1.1.10 #define QAPI_NET_DNS_V6_PRIMARY_SERVER_ID 2

DNS IPv6 primary server ID.

5.1.1.11 #define QAPI_NET_DNS_V6_SECONDARY_SERVER_ID 3

DNS IPv6 secondary server ID.

5.1.1.12 #define gethostbyname(*name*) qapi_Net_DNSc_Get_Host_By_Name(- name)

Macro that returns a pointer to the hostent structure of a host with the given name.

5.1.1.13 #define gethostbyname2(*name*, *af*) qapi_Net_DNSc_Get_Host_By_Name2(name, af)

Macro that returns a pointer to a hostent struct of a host with the given name and family(v4 or v6).

FIBOCOM
Confidential

5.1.2 Data Structure

5.1.2.1 struct qapi_Net_DNS_Server_List_t

Use with [qapi_Net_DNSc_Get_Server_List\(\)](#) to get IP addresses of DNS servers.

Data fields

Type	Parameter	Description
struct ip46addr	svr	DNS servers IP addresses.

5.1.2.2 struct qapi_hostent_s

Data structure returned from [qapi_gethostbyname\(\)](#) or [qapi_gethostbyname2\(\)](#). Same as the UNIX struct `hostent{}`.

Data fields

Type	Parameter	Description
char *	h_name	Official name of the host.
char **	h_aliases	Alias list.
int	h_addrtype	Host address type.
int	h_length	Length of the address.
char **	h_addr_list	List of addresses.

5.1.3 Data Enumeration

5.1.3.1 enum qapi_Net_DNS_Command_t

Commands to start/stop/disable a DNS client.

Enumerator:

QAPI_NET_DNS_DISABLE_E Functionality is deprecated. Do not use.

QAPI_NET_DNS_START_E Allocate space for internal data structures when called for the first time.

For subsequent calls, increases the ref count; DNS query is allowed after the start command. Processes DNS responses from the server.

QAPI_NET_DNS_STOP_E Stop sending DNS requests and processing DNS responses; keeps internal data structures. Frees the space for internal data structures only when the ref count reaches 0.

5.2 API Functions

5.2.1 qapi_Net_DNSc_Set_Client_timeout

Sets the DNSc client timeout value.

Prototype

```
qapi_Status_t qapi_Net_DNSc_Set_Client_timeout ( uint32_t timeout_ms )
```

Parameters

in	<i>timeout_ms</i>	Timeout value in milliseconds.
----	-------------------	--------------------------------

Returns

On success, QAPI_OK.

On error, QAPI_ERROR.

5.2.2 qapi_Net_DNSc_Is_Started

Checks whether the DNS client has started.

Prototype

```
int32_t qapi_Net_DNSc_Is_Started ( void )
```

Returns

0 - Not started.

1 - Started.

5.2.3 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

in	<i>cmd</i>	Command to start/stop/disable the DNS client. The supported commands are QAPI_NET_DNS_START_E, QAPI_NET_DNS_STOP_E, QAPI_NET_DNS_DISABLE_E.
----	------------	--

Returns

On success, 0.

On error, -1.

5.2.4 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

in	<i>hostname</i>	Pointer to an IP address string or host name string.
in	<i>ipaddr</i>	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.

Returns

On success, 0.

On error, <0.

5.2.5 qapi_Net_DNSc_Reshost_on_iface

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

Prototype

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

Parameters

in	<i>hostname</i>	Pointer to an IP address string or host name string.
in	<i>addr</i>	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 addr type is AF_INET, resolve to an IPv4 address. If addr type is AF_INET6, resolve to an IPv6 address.
in	<i>iface_index</i>	Name of the PDN/APN for which the address text string is to be resolved.

Returns

On success, 0.

On error, <0.

5.2.6 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

in	<i>svr_list</i>	Pointer to a buffer to contain the list.
in	<i>iface_index</i>	Index of the configured DNS servers.

Returns

On success, 0.

On error, -1.

5.2.7 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

in	<i>svr_addr</i>	Pointer to the DNS server's IP address string.
in	<i>id</i>	Pointer to the server index. This is filled with the position at which svr_addr is added.
in	<i>iface</i>	Pointer to the interface string on which the server is added.

Returns

On success, QAPI_OK.

On error, QAPI_ERROR.



5.2.8 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

in	<i>svr_addr</i>	Pointer to the DNS server's IP address string.
in	<i>id</i>	Server ID, can be any of the following: 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_SECONDARY_SERVER_ID QAPI_NET_DNS_ANY_SERVER_ID

Returns

On success, 0.

On error, -1.

5.2.9 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

in	<i>svr_addr</i>	Pointer to DNS server's IP address string.
in	<i>id</i>	Server ID, can be any of the following: 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_SECONDARY_SERVER_ID QAPI_NET_DNS_ANY_SERVER_ID
in	<i>iface</i>	Pointer to the name of the PDN on which the server is to be added.

Returns

On success, 0.

On error, -1.

5.2.10 qapi_Net_DNSc_Del_Server

Removes a DNS server from the system.

Prototype

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

Parameters

in	id	Server ID, can be any of the following: 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_SECONDARY_SERVER_ID, or QAPI_NET_DNS_ANY_SERVER_ID

Returns

On success, 0.

On error, -1.

5.2.11 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

in	<i>id</i>	Server ID, can be any of the following: 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_SECONDARY_SERVER_ID QAPI_NET_DNS_ANY_SERVER_ID
in	<i>iface_index</i>	Name of interface from which to delete a DNS server.

Returns

On success, 0.

On error, -1

5.2.12 qapi_Net_DNSc_Host_By_Name

Gets host information for an IPv4 host by name.

Implements a standard Unix version of [gethostbyname\(\)](#). The returned structure should not be freed by the caller.

Prototype

```
qapi_Status_t qapi_Net_DNSc_Host_By_Name ( char * name, struct qapi_hostent_s * ipaddr )
```

Parameters

in	<i>name</i>	Pointer to either a host name or an IPv4 address in standard dot notation.
out	<i>ipaddr</i>	Resolved host information.

Returns

On success, a pointer to a hostent structure. On error, NULL.

5.2.13 qapi_Net_DNSc_Host_By_Name2

Gets host information for an IPv4/IPv6 host by name.

Implements a standard Unix version of [gethostbyname2\(\)](#). The returned hostent structure is not thread safe. It can be freed by internal DNS client routines if the entry ages out or if the table becomes full and space is needed for another entry.

Prototype

```
qapi_Status_t qapi_Net_DNSc_Host_By_Name2 ( char * name, int32_t af, struct qapi_hostent_s *
ipaddr )
```

Parameters

in	<i>name</i>	Pointer to either a host name, an IPv4 address in standard dot notation, or an IPv6 address in colon notation.
in	<i>af</i>	Address family, either AF_INET or AF_INET6.
out	<i>ipaddr</i>	Resolved host information.

Returns

On success, a pointer to a hostent structure.

On error, NULL.

Fibocom

6 MQTT APIs

This chapter describes the MQTT 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`
- `qapi_Net_MQTT_Allow_Unsub_Publish`
- `qapi_Net_MQTT_Set_Extended_Config_Option`
- `qapi_Net_MQTT_Deserialize_Publish_Header`

6.1 MQTT Data Types

MQTT APIs

Net MQTT Length Defines

- #define QAPI_NET_MQTT_MAX_CLIENT_ID_LEN 128
- #define QAPI_NET_MQTT_MAX_LENGTH 65535
- #define QAPI_NET_MQTT_MAX_TOPIC_LEN QAPI_NET_MQTT_MAX_LENGTH
- #define QAPI_NET_MQTT_MAX_WILL_TOPIC_LEN QAPI_NET_MQTT_MAX_LENGTH
- #define QAPI_NET_MQTT_MAX_WILL_MESSAGE_LEN QAPI_NET_MQTT_MAX_LENGTH
- #define QAPI_NET_MQTT_MAX_USERNAME_LEN QAPI_NET_MQTT_MAX_LENGTH
- #define QAPI_NET_MQTT_MAX_PASSWORD_LEN QAPI_NET_MQTT_MAX_LENGTH

6.1.1 Data Define

6.1.1.1 #define QAPI_NET_MQTT_MAX_CLIENT_ID_LEN 128

Maximum client ID length. The MQTT stack uses the same value.

6.1.1.2 #define QAPI_NET_MQTT_MAX_LENGTH 65535

Maximum length, per MQTT specification.

6.1.1.3 #define QAPI_NET_MQTT_MAX_TOPIC_LEN QAPI_NET_MQTT_MAX_LENGTH

Maximum topic length.

6.1.1.4 #define QAPI_NET_MQTT_MAX_WILL_TOPIC_LEN QAPI_NET_MQTT_MAX_L- LENGTH

Maximum will topic length.

6.1.1.5 #define QAPI_NET_MQTT_MAX_WILL_MESSAGE_LEN QAPI_NET_MQTT_MAX_LENGTH

Maximum will message length.

6.1.1.6 #define QAPI_NET_MQTT_MAX_USERNAME_LEN QAPI_NET_MQTT_MAX_LENGTH

Maximum username length.

6.1.1.7 #define QAPI_NET_MQTT_MAX_PASSWORD_LEN QAPI_NET_MQTT_MAX_L- LENGTH

Maximum password length.

6.1.1.8 #define qapi_Net_MQTT_Pass_Pool_Ptr(a, b) mqtt_set_byte_pool(a,b)

Macro that passes Byte Pool Pointer for MQTT Application. Parameter 'a': Handle.

Parameter 'b': Pointer to Byte Pool.

On success, QAPI_OK is returned.

On error, QAPI_ERROR is returned.



Note: This macro is only used in DAM Space.

6.1.1.9 #define qapi_Net_MQTT_Destroy(a) mqtt_destroy_indirection(a)

Macro that releases Byte Pool Pointer for MQTT Application. Parameter 'a' : Handle.

On success, QAPI_OK is returned.

On error, QAPI_ERROR is returned.



Note:

This macro is only used in DAM Space.

6.1.2 Data Structure

6.1.2.1 struct qapi_Net_MQTT_Sock_Opts_t

MQTT socket options.

Data fields

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.

6.1.2.2 struct qapi_Net_MQTT_config_s

MQTT configuration.

Data fields

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 client ID.
int32_t	client_id_len	MQTT client ID length.
uint32_t	keepalive_ duration	Connection 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.
uint32_t	will_topic_len	Will topic length.
uint8_t *	will_message	Will message.
uint32_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.
uint32_t	username_len	Client user length.
uint8_t *	password	Client password.
uint32_t	password_len	Client password length.
uint32_t	connack_timeout_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 certificate details.
qapi_Net_SSL_Cert_t	cert	SSL certificate details.

6.1.2.3 struct qapi_Net_Mqtt_Extended_Config_t

MQTT extended configuration.

Data fields

Reproduction forbidden without Fibocom Wireless Inc. written authorization - All Rights Reserved.

Type	Parameter	Description
uint8_t	options	
qapi_Net_Mqtt_Sec_Mode	sec_mode	MQTT security mode.
qapi_Net_SSL_PSKTable_t	psk	MQTT security mode.
int	pass_msg_hdr	Pass message header. If set, whole publish packet along with MQTT header is passed to MQTT client.
uint16_t	max_chunk	Maximum send data chunk per transaction.
uint16_t	max_chunk_delay_ms	Maximum delay between send data chunks (msec).
uint8_t	max_chunk_retries	Maximum send data chunk retries.

6.1.2.4 struct qapi_Net_MQTT_Pub_config_s

Structure for MQTT publish message.

Data fields

Type	Parameter	Description
int	dup	DUP flag.
int	qos	QOS.
int	retained	Retain flag.
uint16_t	id	Message ID.
uint8_t *	payload	MQTT message payload.
int	payload_length	MQTT message payload length.
int	total_len	Total length of the publish message.

6.1.2.5 struct qapi_Net_MQTTLenString_t

Structure for topic string length in publish message.

Data fields

Type	Parameter	Description
int	len	-
char *	data	-

6.1.2.6 struct qapi_Net_MQTTString_t

Structure for topic in publish message.

Data fields

Type	Parameter	Description
char *	cstring	-
qapi_Net_MQTTLenString_t	lenstring	-

6.1.3 Data Typedef

6.1.3.1 typedef void *qapi_Net_MQTT_Hndl_t

MQTT handle.

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

Connect callback typedef.

6.1.3.3 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.

6.1.3.4 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.

6.1.3.5 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.

6.1.4 Data Enumeration

Reason codes for a subscription callback.

Enumerator:

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.

6.1.4.1 enum QAPI_NET_MQTT_CONNECT_CBK_MSG

Connection callback messages.

Enumerator:

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.

QAPI_NET_MQTT_CONNECT_PROTOCOL_VER_ERROR_E Connection refused -Unacceptable protocol version.

QAPI_NET_MQTT_CONNECT_IDENTIFIER_ERROR_E Connection refused - Identifier rejected.

QAPI_NET_MQTT_CONNECT_SERVER_UNAVAILABLE_E Connection refused - Server unavailable.

QAPI_NET_MQTT_CONNECT_MALFORMED_DATA_E Connection refused - Data in username or password is malformed.

QAPI_NET_MQTT_CONNECT_UNAUTHORIZED_CLIENT_E Connection refused - Unauthorized client.

6.1.4.2 enum QAPI_NET_MQTT_CONN_STATE

Connection states.

Enumerator:

QAPI_NET_MQTT_ST_DORMANT_E Connection is idle.

QAPI_NET_MQTT_ST_TCP_CONNECTING_E TCP is connecting.

QAPI_NET_MQTT_ST_TCP_CONNECTED_E TCP is connected.

QAPI_NET_MQTT_ST_SSL_CONNECTING_E SSL is connecting.

QAPI_NET_MQTT_ST_SSL_CONNECTED_E SSL is connected.



QAPI_NET_MQTT_ST_MQTT_CONNECTING_E MQTT is connecting.

QAPI_NET_MQTT_ST_MQTT_CONNECTED_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.

6.1.4.3 enum QAPI_NET_MQTT_MSG_TYPES

MQTT message types.

Enumerator:

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.

Bitmap for set options.

6.1.4.4 enum qapi_Net_Mqtt_Sec_Mode

MQTT security mode.

Enumerator:

QAPI_MQTT_NO_SECURITY_MODE *No security mode.*

QAPI_MQTT_PSK *PSK mode.*

QAPI_MQTT_CERTIFICATE *Certificate mode.*

6.1.4.5 enum qapi_Net_Mqtt_Extended_Config_Options_t

MQTT extended configuration options.

Enumerator:

QAPI_NET_MQTT_EXTENDED_CONFIG_SEC_MODE
QAPI_NET_MQTT_EXTENDED_CONFIG_PSK_TABLE_T
QAPI_NET_MQTT_EXTENDED_CONFIG_PASS_MSG_HEADER
QAPI_NET_MQTT_EXTENDED_CONFIG_MAX_CHUNK
QAPI_NET_MQTT_EXTENDED_CONFIG_MAX_CHUNK_DELAY
QAPI_NET_MQTT_EXTENDED_CONFIG_MAX_CHUNK_RETRIES
QAPI_NET_MQTT_EXTENDED_CONFIG_MAX

6.2 API Functions

6.2.1 qapi_Net_MQTT_New

Creates a new MQTT context.

Prototype

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

Parameters

out	<i>hndl</i>	Newly created MQTT context.
-----	-------------	-----------------------------

Returns

QAPI_OK on success or QAPI_ERROR on failure.

6.2.2 qapi_Net_MQTT_Destroy

Destroys an MQTT context.

Prototype

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

Parameters

in	<i>hndl</i>	Handle for the MQTT context to be destroyed.
----	-------------	--

Returns

QAPI_OK on success or QAPI_ERROR on failure.

6.2.3 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

in	<i>hndl</i>	MQTT handle.
in	<i>config</i>	MQTT client configuration.

Returns

QAPI_OK on success or < 0 on failure.

6.2.4 qapi_Net_MQTT_Disconnect

Disconnects from an MQTT broker.

Prototype

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

Parameters

in	<i>hndl</i>	MQTT handle.
----	-------------	--------------

Returns

QAPI_OK on success or < 0 on failure.

6.2.5 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

in	hndl	MQTT handle.
in	topic	MQTT topic.
in	msg	MQTT payload.
in	msg_len	MQTT payload length.
in	qos	QOS to be used for the message.
in	retain	Retain flag.

Returns

QAPI_OK on success or < 0 on failure.

6.2.6 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_thndl, const uint8_t *
topic, const uint8_t * msg, int32_t msg_len, int32_t qos, bool retain, uint16_t * msg_id )
```

Parameters

in	hndl	MQTT handle.
in	topic	MQTT topic.
in	msg	MQTT payload.
in	msg_len	MQTT payload length.
in	qos	QoS to be used for the message.
in	retain	Retain flag.
out	msg_id	Message ID of the MQTT publish message.

Returns

QAPI_OK on success or < 0 on failure.

6.2.7 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

in	<i>hndl</i>	MQTT handle.
in	<i>topic</i>	Subscription topic.
in	<i>qos</i>	QoS to be used.

Returns

QAPI_OK on success or < 0 on failure.

6.2.8 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

in	<i>hndl</i>	MQTT handle
in	<i>topic</i>	Topic from which to unsubscribe.

Returns

QAPI_OK on success or < 0 on failure.

6.2.9 qapi_Net_MQTT_Set_Connect_Callback

Sets a connect callback, which is invoked when the 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

in	<i>hndl</i>	MQTT handle.
in	<i>callback</i>	Callback to be invoked.

Returns

Success or failure.

6.2.10qapi_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

in	<i>hndl</i>	MQTT handle.
in	<i>callback</i>	Callback to be invoked.

Returns

QAPI_OK on success or < 0 on failure.

6.2.11 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

in	<i>hndl</i>	MQTT handle.
in	<i>callback</i>	Callback to be invoked.

Returns

QAPI_OK on success or < 0 on failure.

6.2.12qapi_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

in	<i>hndl</i>	MQTT handle.
in	<i>callback</i>	Callback to be invoked.

Returns

QAPI_OK on success or < 0 on failure.

6.2.13qapi_Net_MQTT_Allow_Unsub_Publish

Sets an unsubscribe callback, which will allow messages to be received for an unsubscribed topic.

Prototype

```
qapi_Status_t qapi_Net_MQTT_Allow_Unsub_Publish ( qapi_Net_MQTT_Hndl_t hndl, bool
allow_unsub_pub )
```

Parameters

in	<i>hndl</i>	MQTT handle.
in	<i>allow_unsub_pub</i>	Condition that allows this behavior.

Returns

QAPI_OK on success or < 0 on failure.

6.2.14 qapi_Net_MQTT_Set_Extended_Config_Option

Sets the extended configuration parameters. It is expected that the user will call this API before these parameters will be used in another API.

Prototype

```
qapi_Status_t qapi_Net_MQTT_Set_Extended_Config_Option ( qapi_Net_MQTT_Hndl_t hndl,
qapi_Net_Mqtt_Extended_Config_Options_t ext_option, void * val, int32_t val_len )
```

Parameters

in	<i>hndl</i>	MQTT handle.
in	<i>ext_option</i>	Option type to be set.
in	<i>val</i>	Values to be set.
in	<i>val_len</i>	Values length.

Returns

QAPI_OK on success or < 0 on failure.

6.2.15qapi_Net_MQTT_Deserialize_Publish_Header

Gets the publish message along with the MQTT header.

Prototype

```
qapi_Status_t qapi_Net_MQTT_Deserialize_Publish_Header ( qapi_Net_MQTT_Hndl_t hndl, unsigned
char * buf, int buf_len, qapi_Net_MQTTString_t *topic_n, qapi_Net_MQTT_Pub_Config_t * pub )
```

Parameters

in	<i>hndl</i>	MQTT handle.
in	<i>buf</i>	The raw buffer data.
in	<i>buf_len</i>	The length in bytes of the data in the supplied buffer.
out	<i>topic_n</i>	Returned MQTTString, the MQTT topic in the publish.
out	<i>pub</i>	Returned value containing the DUP flag, QOS value, retain flag, packet identifier, publish payload, length of the publish payload.

Returns

QAPI_OK on success or < 0 on failure.

FIBOCOM
Confidential

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.

- [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](#)
- [qapi_Net_HTTPc_Extended_Config_Options](#)

7.1 HTTP(S) Data Types

7.1.1 Data Define

7.1.1.1 #define qapi_Net_HTTPc_Free_sess(a) httpc_destroy_indirection(a, TXM_QAPI_HTTPC_FREE_SESSION)

Macro that releases Byte Pool Pointer for HTTP Client. Parameter 'a' : Handle.

On success, QAPI_OK is returned.

On error, QAPI_ERROR is returned.



Note:

This macro is only used in DAM Space.

7.1.1.2 #define qapi_Net_HTTPc_Pass_Pool_Ptr(a, b) httpc_set_byte_pool(a,b)

Macro that passes Byte Pool Pointer for HTTP Client. Parameter 'a': Handle.

Parameter 'b': Pointer to Byte Pool.

On success, QAPI_OK is returned.

On error, QAPI_ERROR is returned.



Note:

This macro is only used in DAM Space.

7.1.2 Data Structure

7.1.2.1 struct qapi_Net_HTTPc_Response_t

HTTP response data returned by [qapi_HTTPc_CB_t\(\)](#).

Data fields

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.

7.1.2.2 struct qapi_Net_HTTPc_Sock_Opts_s

HTTP socket options.

Data fields

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.

7.1.2.3 struct qapi_Net_HTTPc_Config_t

Structure to configure an HTTP client session.

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

7.1.3 Data Typedef

7.1.3.1 typedef void(* qapi_HTTPc_CB_t)(void *arg, int32_t state, void *value)

HTTP response user callback registered during [qapi_Net_HTTPc_New_sess\(\)](#).

Parameters

in	<i>arg</i>	User payload information.
in	<i>state</i>	HTTP response state.
in	<i>value</i>	HTTP response information.

7.1.3.2 typedef void* qapi_Net_HTTPc_handle_t

HTTP client handle used by most of [qapi_Net_HTTPc_xxxx\(\)](#).

7.1.4 Data Enumeration

7.1.4.1 enum qapi_Net_HTTPc_Method_e

HTTP request types supported by [qapi_Net_HTTPc_Request\(\)](#).

Enumerator:

QAPI_NET_HTTP_CLIENT_GET_E	<i>HTTP get request</i>
QAPI_NET_HTTP_CLIENT_POST_E	<i>HTTP post request</i>
QAPI_NET_HTTP_CLIENT_PUT_E	<i>HTTP put request</i>
QAPI_NET_HTTP_CLIENT_PATCH_E	<i>HTTP patch request</i>
QAPI_NET_HTTP_CLIENT_HEAD_E	<i>HTTP head request</i>
QAPI_NET_HTTP_CLIENT_CONNECT_E	<i>HTTP connect request</i>
QAPI_NET_HTTP_CLIENT_DELETE_E	<i>HTTP delete request</i>

7.1.4.2 enum qapi_Net_HTTPc_CB_State_e

HTTP callback state returned by [qapi_HTTPc_CB_t\(\)](#).

Enumerator:

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.

7.1.4.3 enum qapi_Net_HTTPc_Extended_Config_Options_e

HTTP extended configuration options supported by [qapi_Net_HTTPc_Extended_Config_Options\(\)](#).

Enumerator:

QAPI_NET_HTTPC_EXTENDED_CONFIG_NONE Extended config option – not valid

QAPI_NET_HTTPC_EXTENDED_CONFIG_DNS_RESOLVE_INTERFACE Extended config option –
DSN resolve hostname on a specific interface

QAPI_NET_HTTPC_EXTENDED_CONFIG_BIND_ADDRESS Extended config option – send data on a
specific interface

7.1 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 )
```

Returns

On success, 0 is returned. Other value on error.

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 )
```

Returns

On success, 0 is returned. Other value on error.

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. Any HTTP connection or security related configuration can be configured using the configuration QAPIs.

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

in	<i>timeout</i>	Timeout (in ms) of a session method (zero is not recommended).
in	<i>ssl_Object_Handle</i>	SSL context for HTTPs connect (zero for no HTTPs session support).
in	<i>callback</i>	Register a callback function; NULL for no support for a callback.
in	<i>arg</i>	User data payload to be returned by the callback function.
in	<i>httpc_Max_Body_Length</i>	Maximum body length for this session.
in	<i>httpc_Max_Header_Length</i>	Maximum header length for this session.

Returns

On success, qapi_Net_HTTPc_handle_t is returned. NULL otherwise.

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

in	<i>handle</i>	Handle to the HTTP client session.
----	---------------	------------------------------------

Returns

On success, 0 is returned. Other value on error.

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

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>URL</i>	Server URL information.
in	<i>port</i>	Server port information.

Returns

On success, 0 is returned. Other value on error.

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_thandle, const char * URL,  
uint16_t port, uint8_t secure_proxy )
```

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>URL</i>	Server URL information.
in	<i>port</i>	Server port information.
in	<i>secure_proxy</i>	Secure proxy connection.

Returns

On success, 0 is returned. Other value on error.

7.2.7 qapi_Net_HTTPc_Disconnect

Disconnects an HTTP client session from the HTTP server.

The HTTP client session that is connected to the HTTP server is disconnected from the HTTP server.

Prototype

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

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
----	---------------	------------------------------------

Returns

On success, 0 is returned. Other value on error.

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

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>cmd</i>	HTTP request method information.
in	<i>URL</i>	Server URL information.

Returns

On success, 0 is returned. Other value on error.

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. Maximum allowed body size is configured at HTTP session creation. Setting body length to zero would clear the HTTP body.

Prototype

```
qapi_Status_t qapi_Net_HTTPc_Set_Body ( qapi_Net_HTTPc_handle_thandle, const char * body,
uint32_t body_Length )
```

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>body</i>	HTTP body related information buffer.
in	<i>body_Length</i>	HTTP body buffer length.

Returns

On success, 0 is returned. Other value on error.

7.2.10qapi_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_thandle, const char * key, const char * value )
```

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>key</i>	HTTP key related information.
in	<i>value</i>	HTTP value associated with the key.

Returns

On success, 0 is returned. Other value on error.

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. Maximum allowed header size is configured at HTTP session creation.

Prototype

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

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>type</i>	HTTP header type related information.
in	<i>value</i>	HTTP value associated with the header type.

Returns

On success, 0 is returned. Other value on error.

7.2.12qapi_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

in	<i>handle</i>	Handle to the HTTP client session.
----	---------------	------------------------------------

Returns

On success, 0 is returned. Other value on error.

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_t qapi_Net_HTTPc_Configure_SSL ( qapi_Net_HTTPc_handle_thandle,  
qapi_Net_SSL_Config_t * ssl_Cfg )
```

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>ssl_Cfg</i>	SSL configuration information.

Returns

On success, 0 is returned. Other value on error.

7.2.14qapi_Net_HTTPc_Configure

Configures the HTTP client session based on the application requirement.

Prototype

```
qapi_Status_t qapi_Net_HTTPc_Configure ( qapi_Net_HTTPc_handle_thandle,  
qapi_Net_HTTPc_Config_t * httpc_Cfg )
```

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>httpc_Cfg</i>	HTTP client configuration information.

Returns

On success, 0 is returned. Other value on error.

7.2.15qapi_Net_HTTPc_Extended_Config_Options

Extended configuration options for HTTP client session based on the application requirement.

Prototype

```
qapi_Status_t qapi_Net_HTTPc_Extended_Config_Options ( qapi_Net_HTTPc_handle_t handle,
qapi_Net_HTTPc_Extended_Config_Options_e option,void * option_value, uint32_t option_size )
```

Parameters

in	<i>handle</i>	Handle to the HTTP client session.
in	<i>option</i>	HTTP client extended configuration option.
in	<i>option_value</i>	HTTP client extended configuration option info.
in	<i>option_size</i>	HTTP client extended configuration option size.

Returns

On success, 0. Returns any other value on error.

8 Device Information Module

This chapter describes the device information data types and APIs.

- [qapi_Device_Info_Init_v2](#)
- [qapi_Device_Info_Get_v2](#)
- [qapi_Device_Info_Set_Callback_v2](#)
- [qapi_Device_Info_Release_v2](#)
- [qapi_Device_Info_Reset_v2](#)
- [qapi_Device_Info_Request](#)
- [qapi_Device_Info_Clear_Callback_v2](#)

FIBOCOM
Confidential

8.1 Device Information

8.1.1 Data Define

8.1.1.1 #define QAPI_DEVICE_INFO_MODE_PREF_GSM 0x04

Mode preference: GSM.

8.1.1.2 #define QAPI_DEVICE_INFO_MODE_PREF_UMTS 0x08

Mode preference: UMTS.

8.1.1.3 #define QAPI_DEVICE_INFO_MODE_PREF_LTE 0x10

Mode preference: LTE.

8.1.1.4 #define QAPI_DEVICE_INFO_LTE_OP_MODE_PREF_WB ((uint64_t)0x01)

Macros for ciot_mode_preference(QAPI_DEVICE_INFO_CIoT_LTE_OP_MODE_PREF_E). LTE wideband.

8.1.1.5 #define QAPI_DEVICE_INFO_LTE_OP_MODE_PREF_M1 ((uint64_t)0x02)

LTE M1.

8.1.1.6 #define QAPI_DEVICE_INFO_LTE_OP_MODE_PREF_NB1 ((uint64_t)0x04)

LTE NB1.

8.1.1.7 #define QAPI_CELL_SEARCH_MAX_NUM 8

Maximum number of PCIs per RAT.

8.1.1.8 #define QAPI_PCI_SCAN_MAX_NUM_PLMN 6

Maximum number of MNC/MCC for a single cell.

**8.1.1.9 #define QAPI_DEVICE_INFO_OPERATING_BAND_1
((uint64_t)0x0000000000000001ull)**

Macros for band preference. E-UTRA Operating Band 1.

**8.1.1.10 #define QAPI_DEVICE_INFO_OPERATING_BAND_2
((uint64_t)0x0000000000000002ull)**

E-UTRA Operating Band 2.

**8.1.1.11 #define QAPI_DEVICE_INFO_OPERATING_BAND_3
((uint64_t)0x0000000000000004ull)**

E-UTRA Operating Band 3.

**8.1.1.12 #define QAPI_DEVICE_INFO_OPERATING_BAND_5
((uint64_t)0x0000000000000010ull)**

E-UTRA Operating Band 5.

**8.1.1.13 #define QAPI_DEVICE_INFO_OPERATING_BAND_6
((uint64_t)0x0000000000000020ull)**

E-UTRA Operating Band 6.

8.1.1.14 #define QAPI_DEVICE_INFO_OPERATING_BAND_7
((uint64_t)0x0000000000000040ull)

E-UTRA Operating Band 7.

8.1.1.15 #define QAPI_DEVICE_INFO_OPERATING_BAND_8
((uint64_t)0x0000000000000080ull)

E-UTRA Operating Band 8.

8.1.1.16 #define QAPI_DEVICE_INFO_OPERATING_BAND_9
((uint64_t)0x0000000000000100ull)

E-UTRA Operating Band 9.

8.1.1.17 #define QAPI_DEVICE_INFO_OPERATING_BAND_10 **((uint64_t)0x0000000000000200ull)**

E-UTRA Operating Band 10.

8.1.1.18 #define QAPI_DEVICE_INFO_OPERATING_BAND_11 **((uint64_t)0x0000000000000400ull)**

E-UTRA Operating Band 11.

8.1.1.19 #define QAPI_DEVICE_INFO_OPERATING_BAND_12 **((uint64_t)0x0000000000000800ull)**

E-UTRA Operating Band 12.

8.1.1.20 #define QAPI_DEVICE_INFO_OPERATING_BAND_13 ((uint64_t)0x000000000001000ull)

E-UTRA Operating Band 13.

8.1.1.21 #define QAPI_DEVICE_INFO_OPERATING_BAND_14 ((uint64_t)0x000000000002000ull)

E-UTRA Operating Band 14.

8.1.1.22 #define QAPI_DEVICE_INFO_OPERATING_BAND_17 ((uint64_t)0x0000000000010000ull)

E-UTRA Operating Band 17.

8.1.1.23 #define QAPI_DEVICE_INFO_OPERATING_BAND_18 ((uint64_t)0x0000000000020000ull)

E-UTRA Operating Band 18.

8.1.1.24 #define QAPI_DEVICE_INFO_OPERATING_BAND_19 ((uint64_t)0x0000000000040000ull)

E-UTRA Operating Band 19.

8.1.1.25 #define QAPI_DEVICE_INFO_OPERATING_BAND_20 ((uint64_t)0x0000000000080000ull)

E-UTRA Operating Band 20

8.1.1.26 #define QAPI_DEVICE_INFO_OPERATING_BAND_21 ((uint64_t)0x00000000000100000ull)

E-UTRA Operating Band 21.

8.1.1.27 #define QAPI_DEVICE_INFO_OPERATING_BAND_23 ((uint64_t)0x00000000000400000ull)

E-UTRA Operating Band 23.

8.1.1.28 #define QAPI_DEVICE_INFO_OPERATING_BAND_24 ((uint64_t)0x000000000800000ull)

E-UTRA Operating Band 24.

8.1.1.29 #define QAPI_DEVICE_INFO_OPERATING_BAND_25 ((uint64_t)0x000000001000000ull)

E-UTRA Operating Band 25.

8.1.1.30 #define QAPI_DEVICE_INFO_OPERATING_BAND_26 ((uint64_t)0x000000002000000ull)

E-UTRA Operating Band 26.

8.1.1.31 #define QAPI_DEVICE_INFO_OPERATING_BAND_28 ((uint64_t)0x000000008000000ull)

E-UTRA Operating Band 28.

8.1.1.32 #define QAPI_DEVICE_INFO_OPERATING_BAND_29 ((uint64_t)0x000000010000000ull)

E-UTRA Operating Band 29.

8.1.1.33 #define QAPI_DEVICE_INFO_OPERATING_BAND_32 ((uint64_t)0x000000020000000ull)

E-UTRA Operating Band 32.

8.1.1.34 #define QAPI_DEVICE_INFO_OPERATING_BAND_30 ((uint64_t)0x000000080000000ull)

E-UTRA Operating Band 30.

8.1.1.35 #define QAPI_DEVICE_INFO_OPERATING_BAND_33 ((uint64_t)0x000000100000000ull)

**8.1.1.36 #define QAPI_DEVICE_INFO_OPERATING_BAND_34 ((uint64_-
t)0x000000200000000ull)**

E-UTRA Operating Band 34.

**8.1.1.37 #define QAPI_DEVICE_INFO_OPERATING_BAND_35 ((uint64_-
t)0x000000400000000ull)**

E-UTRA Operating Band 35.

**8.1.1.38 #define QAPI_DEVICE_INFO_OPERATING_BAND_36 ((uint64_-
t)0x000000800000000ull)**

E-UTRA Operating Band 36.

**8.1.1.39 #define QAPI_DEVICE_INFO_OPERATING_BAND_37 ((uint64_-
t)0x000001000000000ull)**

E-UTRA Operating Band 37.

**8.1.1.40 #define QAPI_DEVICE_INFO_OPERATING_BAND_38 ((uint64_-
t)0x000002000000000ull)**

E-UTRA Operating Band 38.

**8.1.1.41 #define QAPI_DEVICE_INFO_OPERATING_BAND_39 ((uint64_-
t)0x000004000000000ull)**

E-UTRA Operating Band 39.

**8.1.1.42 #define QAPI_DEVICE_INFO_OPERATING_BAND_40 ((uint64_-
t)0x000008000000000ull)**

E-UTRA Operating Band 40.

8.1.1.43 #define QAPI_DEVICE_INFO_OPERATING_BAND_41 ((uint64_t)0x0000100000000000ull)

E-UTRA Operating Band 41.

8.1.1.44 #define QAPI_DEVICE_INFO_OPERATING_BAND_42 ((uint64_t)0x0000200000000000ull)

E-UTRA Operating Band 42.

8.1.1.45 #define QAPI_DEVICE_INFO_OPERATING_BAND_43 ((uint64_t)0x0000400000000000ull)

E-UTRA Operating Band 43.

8.1.1.46 #define QAPI_DEVICE_INFO_OPERATING_BAND_46 ((uint64_t)0x0002000000000000ull)

E-UTRA Operating Band 46.

8.1.1.47 #define QAPI_DEVICE_INFO_OPERATING_BAND_47 ((uint64_t)0x0004000000000000ull)

E-UTRA Operating Band 47.

8.1.1.48 #define QAPI_DEVICE_INFO_OPERATING_BAND_48 ((uint64_t)0x0008000000000000ull)

E-UTRA Operating Band 48.

8.1.1.49 #define QAPI_DEVICE_INFO_OPERATING_BAND_125 ((uint64_t)0x10000000000000000ull)

E-UTRA Operating Band 125.

8.1.1.50 #define QAPI_DEVICE_INFO_OPERATING_BAND_126 ((uint64_t)0x2000000000000000ull)

E-UTRA Operating Band 126.

8.1.1.51 #define QAPI_DEVICE_INFO_OPERATING_BAND_127 ((uint64_t)0x4000000000000000ull)

E-UTRA Operating Band 127.

8.1.1.52 #define QAPI_DEVICE_INFO_SRV_TYPE_LTE 1

Macro for Network Bearer values (QAPI_DEVICE_INFO_NETWORK_BEARER_E). Nw bearer svc type: LTE.

8.1.1.53 #define QAPI_DEVICE_INFO_SRV_TYPE_GSM 2

Nw bearer svc type: GSM.

8.1.1.54 #define QAPI_DEVICE_INFO_SRV_TYPE_WCDMA 3

Nw bearer svc type: WCDMA.

8.1.1.55 #define QAPI_DEVICE_INFO_LTE_TDD 5

Nw bearer svc type: LTE Mode: TDD.

8.1.1.56 #define QAPI_DEVICE_INFO_LTE_FDD 6

Nw bearer svc type: LTE Mode: FDD.

8.1.1.57 #define QAPI_DEVICE_INFO_LTE_NB_IOT 7

Nw bearer svc type: LTE Mode: NB-IOT.

8.1.1.58 #define QAPI_DEVICE_INFO_BUF_SIZE 128

Maximum size of `qapi_Device_Info_t` valuebuf.

8.1.1.59 #define JULIAN_YEAR 1980

Julian year.

8.1.1.60 #define DEVICE_INFO_MAX_INSTANCES 5

Maximum number of device info instances.

8.1.1.61 #define MAX_LEN_VAL 5

Maximum length of the integer array.

8.1.1.62 #define QAPI_DEVICE_INFO_EDRX_VALUE_NOT_USED 0xFF

edrx_value or paging_time_window not used.

8.1.1.63 #define QAPI_DEVICE_INFO_APN_NAME_MAX 150

Maximum APN name length.

8.1.1.64 #define QAPI_DEVICE_INFO_USERNAME 127

Maximum user name length.

8.1.1.65 #define QAPI_DEVICE_INFO_PASSWORD 127

Maximum password length.

8.1.1.66 #define QAPI_MAX_ERR_CODE QAPI_ERROR(QAPI_MOD_BASE,(-16))

General error code maximum value.

8.1.1.67 #define QAPI_PLMN_LIST_MAX_LEN 8

Maximum length of PLMN list.

8.1.1.68 #define qapi_Device_Info_Pass_Pool_Ptr(a, b) device_info_set_byte_pool(a,b)

Macro that passes Byte Pool Pointer for Application invoking device information QAPIs. Parameter 'a': Handle.

Parameter 'b': Pointer to Byte Pool.

On success, QAPI_OK is returned. On error, appropriate QAPI_ERR_code is returned.



Note: This macro is only used in DAM Space.

8.1.2 Data Structure

8.1.2.1 struct qapi_Device_Info_t

QAPI device information structure.

Data fields

Type	Parameter	Description
qapi_Device_	id	Required information ID.

Info_ID_t		
qapi_Device_ - Info_Type_t	info_type	Response type.
union qapi_ - Device_Info_t	u	Union of values.

8.1.2.2 union qapi_Device_Info_t.u

Union of values.

Data fields

Type	Parameter	Description
u	valuebuf	Union of buffer values.
int64_t	valueint	Response integer value.
bool	valuebool	Response Boolean value.
u	valint	Union of integer array values.
void *	device_info_ - extrn	Extended parameter.

8.1.2.3 struct qapi_Device_Info_t.u.valuebuf

Union of buffer values.

Data fields

Type	Parameter	Description
char	buf	Response buffer.
uint32_t	len	Length of the response string.

8.1.2.4 struct qapi_Device_Info_t.u.valint

Union of integer array values.

Data fields

Type	Parameter	Description
int	buf	Response type: integer array.
uint32_t	len	Length of the array.

8.1.2.5 struct qapi_Gsm_Cell_Info

QAPI cell scan information for GSM structure.

Data fields

Type	Parameter	Description
uint16_t	arfcn	Absolute RF channel number.
uint8_t	bsic	Base station identity code.
uint16_t	mcc	A 16-bit integer representation of MCC. Range: 0 to 999.
uint16_t	mnc	A 16-bit integer representation of MNC. Range: 0 to 999.
uint16_t	lac	Location area code.
uint16_t	cell_identity	Cell identity.
boolean	cell_barred	Cell barred.
int16_t	rssi	Cell Rx measurement. Values range between 0 and 63.
int	gprs_support	Informs if PS is supported.

8.1.2.6 struct qapi_Lte_Cell_Info

QAPI cell scan information for LTE structure.

Data fields

Type	Parameter	Description
uint32_t	earfcn	Extended absolute RF channel number.
uint16_t	phy_cell_id	Physical cell ID.
uint32_t	plmn_len	Must be set to number of elements in MCC and MNC.
uint16_t	mcc	List of associated MCC.
uint16_t	mnc	List of associated MNC.
uint16_t	tac	Tracking area code (only applicable for LTE).

uint32_t	cell_id	Global cell ID.
boolean	cell_barred	Cell barred.
int16_t	rsrp	Combined rsrp.
int16_t	rsrq	Combined rsrq.
int	bandwidth	Band indicator for the ARFCN provided.
int16_t	rssi	Combined rssi.

FIBOCOM
Confidential

8.1.2.7 struct qapi_Cell_Info

QAPI cell scan information structure.

Data fields

Type	Parameter	Description
size_t	size	Size of the structure.
uint32_t	cinfo_len	Set to number of elements in cell search information.
qapi_Gsm_Cell_Info	g_cinfo	GERAN cell search information.
qapi_Lte_Cell_Info	lte_cinfo	LTE cell search information.

8.1.2.8 struct qapi_Device_Info_Req_Verify_Pin_t

QAPI verify PIN request structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_Pin_Id_t	pin_id	Pin ID to verify.
char	pin_value	Pin value to verify.

8.1.2.9 struct qapi_Device_Info_Req_Change_Pin_t

QAPI change PIN request structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_Pin_Id_t	pin_id	Pin ID to change.
char	old_pin_value	Old pin value.
char	new_pin_value	New pin value.

8.1.2.10 struct qapi_Device_Info_Req_Unblock_Pin_t

QAPI unblock PIN request structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_Pin_Id_t	pin_id	Pin ID to unblock.
char	puk_value	PUK pin value to unblock PIN.
char	new_pin_value	New pin value.

8.1.2.11 struct qapi_Device_Info_Req_Protect_Pin_t

QAPI protect PIN request structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_Pin_Id_t	pin_id	Pin ID.
char	pin_value	Pin value.
uint8_t	isEnabledPin	Enable/disable PIN operation. TRUE means Enable PIN.

8.1.2.12 struct qapi_Device_Info_Req_Set_FDN_t

QAPI set FDN status request structure.

Data fields

Type	Parameter	Description
uint8_t	isEnabledFDN	Enable/disable FDN.

8.1.2.13 struct qapi_Device_Info_APDU_Command_t

QAPI device information APDU command structure.

Data fields

Type	Parameter	Description
uint8_t	apdu_cmd_length	APDU command data length.
uint8_t *	apdu_cmd_data	APDU command data.

8.1.2.14 struct qapi_Device_Info_UIM_Response_t

QAPI device information UIM response structure.

Data fields

Type	Parameter	Description
------	-----------	-------------

qapi_Device- _Info_UIM_- Response_code	error	UIM response code.
uint8_t	card_result_- sw1	SW1 status word1 received from the card.
uint8_t	card_result_- sw2	SW2 status word2 received from the card.

8.1.2.15 struct qapi_Device_Info_Pin_Response_t

QAPI device information UIM PIN response structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_UIM_- Response_t	uim_response	UIM response data.
uint8_t	retries_left_- valid	Indicates if verify_left/unblock_left are valid.
uint8_t	verify_left	Number of remaining attempts to verify the PIN.
uint8_t	unblock_left	Number of remaining attempts to unblock the PIN.

8.1.2.16 struct qapi_Device_Info_Get_FDN_Response_t

QAPI device information FDN get response structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_FDN_- Status_t	fdn_status	FDN status value.

8.1.2.17 struct qapi_Device_Info_APDU_Response_t

QAPI device information APDU command response structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_UIM_- Response_t	uim_response	UIM response data.
uint8_t	apdu_resp_- length	APDU command response data length.
uint8_t *	apdu_resp_data	APDU command response data.

8.1.2.18 struct qapi_Device_Info_PREFERRED_PLMN_t

QAPI device information Preferred PLMN data structure.

Data fields

Type	Parameter	Description
struct qapi_- Device_Info_- Preferred_PL- MN_s *	next	Pointer to the next preferred PLMN item information.

uint8_t	index	Index ID of PLMN item.
uint8_t	plmn	PLMN code.
uint8_t	gsm_act:1	GSM access technology selected.
uint8_t	gsm_compact_act:1	GSM COMPACT access technology selected.
uint8_t	utran_act:1	UTRAN access technology selected.
uint8_t	eutran_act:1	EUTRAN access technology selected.

8.1.2.19 struct qapi_Device_Info_PREFERRED_PLMN_Set_Req_t

QAPI device information Preferred PLMN set request structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_PREFERRED_PLMN_List_ID_t	list_id	PLMN List type.
qapi_Device_Info_PREFERRED_PLMN_t	plmn_data	Preferred PLMN data item.

8.1.2.20 struct qapi_Device_Info_PREFERRED_PLMN_Read_Req_t

QAPI device information Preferred PLMN read request structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_PREFERRED_PLMN_List_ID_t	list_id	PLMN List type.

8.1.2.21 struct qapi_Device_Info_PREFERRED_PLMN_Read_Rsp_t

QAPI device information Preferred PLMN read response structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_UIM_Response_t	uim_response	UIM response data.
uint8_t	total_plmn_number	Total PLMN item number in PLMN List .
uint8_t	valid_plmn_number	Valid PLMN item number in PLMN List .
qapi_Device_Info_PREFERRED_PLMN_t *	plmn_list_head	Valid PLMN item data list.

8.1.2.22 struct qapi_Device_Info_UICC_Application_Rsp_t

QAPI device information UICC Application read response structure.

Data fields

Type	Parameter	Description
qapi_Device-Info_UIM-Response_t	uim_response	UIM response data.
uint8_t	application_rec	Records data from EFDIR file.
uint8_t	application_rec_number	Number of records in EFDIR file.
uint8_t	application_rec_length	Length of each record in EFDIR file.

8.1.2.23 struct qapi_Device_Info_Req_EDRX_Set_t

EDRX set request parameter, map to [QAPI_DEVICE_INFO_REQ_EDRX_SET_E](#) ID.

Data fields

Type	Parameter	Description
boolean	edrx_mode	0 - disable, 1 - enable.
qapi_Device-Info_EDRX-Act_Type_t	act_type	
uint8_t	edrx_value	Bit-4 to 1 of octet3 of the extended DRX parameter information element. See subclause 10.5.5.32 of 24.008, scope 0 ~ 0x0F.

8.1.2.24 struct qapi_Device_Info_Req_EDRX_Get_t

EDRX gets request parameter and maps to [QAPI_DEVICE_INFO_REQ_EDRX_GET_E](#) ID.

Data fields

Type	Parameter	Description
qapi_Device_Info_EDRX_Act_Type_t	act_type	

8.1.2.25 struct qapi_Device_Info_Rsp_EDRX_Get_t

EDRX gets response parameter and maps to [QAPI_DEVICE_INFO_REQ_EDRX_GET_E](#) ID.

Data fields

Type	Parameter	Description
boolean	edrx_mode	0 - disable, 1 - enable.
qapi_Device_Info_EDRX_Act_Type_t	act_type	
uint8_t	edrx_value	Bit-4 to 1 of octet3 of the extended DRX parameter information element. See subclause 10.5.5.32 of 24.008, scope 0 ~ 0x0F.
uint8_t	paging_time_window	Bit-8 to 5 of octet3 of the extended DRX parameter information element. See subclause 10.5.5.32 of 24.008, scope 0 ~ 0x0F.

8.1.2.26 struct qapi_Device_Info_Jamming_Status_Get_Rsp_t

QAPI device information jamming status gets response structure.

Data fields

Type	Parameter	Description
qapi_Device-Info_RAT_Type_t	rat_type	RAT type.
uint8_t	jammer_status- _valid	Must be set to true if jammer_status is being passed.
uint16_t	jammer_status	Jammer_status.
uint8_t	soft_jammer_- flag_valid	Must be set to true if soft_jammer_flag is being passed.
uint8_t	soft_jammer_- flag	Soft jammer flag.

8.1.2.27 struct qapi_Device_Info_Jamming_Status_Get_Req_t

QAPI device information jamming status gets request structure.

Data fields

Type	Parameter	Description
qapi_Device-Info_RAT_Type_t	rat_type	RAT type.

8.1.2.28 struct qapi_Device_Info_IMS_Registration_Status_Rsp_t

QAPI device information IMS registration response structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_IMS_Reg_Status_t	registration_status	

8.1.2.29 struct qapi_Device_Info_Call_Status_Rsp_t

QAPI device information voice call response structure.

Data fields

Type	Parameter	Description
uint8_t	call_id	Unique call identifier for the call.
uint8_t	call_state_valid	Must be set to TRUE if call_state is being passed.
qapi_Device_Info_Call_State_t	call_state	Call state.
uint8_t	direction_valid	Must be set to TRUE if direction is being passed.
qapi_Device_Info_Call_Direction_t	direction	Call redirection, MO or MT.

8.1.2.30 struct qapi_Device_Info_Scan_Config_Rsp_t

QAPI device info scan configuration response structure.

Type	Parameter	Description
uint8_t	scan_counter_valid	Must be set to TRUE if scan_counter is being passed.
uint8_t	scan_counter	The number of times a device shall scan all rats in the rat priority list before entering power save.
uint8_t	power_save_duration_min_valid	Must be set to TRUE if power_save_duration_min is being passed.
uint32_t	power_save_duration_min	Value in seconds. The initial duration for which the device will stop scanning if no service is found after scanning for a number of attempts defined by scan_counter.
uint8_t	power_save_duration_inc_type_valid	Must be set to TRUE if power_save_duration_inc_type is being passed.
qapi_Device_Info_Power_Save_Duration_Inc_Type_t	power_save_duration_inc_type	Information on how the power duration value increases values: QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_STATIC (0x00) – Power save duration remains static. QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_LINEAR (0x01) – Power save duration gets incremented linearly. QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_EXP (0x02) – Power save duration gets incremented exponentially.
uint8_t	power_save_duration_max_valid	Must be set to TRUE if power_save_duration_max is being passed.
uint32_t	power_save_duration_max	Value in seconds. The maximum duration for which the device will

		stop scanning if no service is found after scanning for a number of attempts defined by scan_counter.
uint8_t	hplmn_scan_interval_valid	Must be set to TRUE if hplmn_scan_interval is being passed.
uint32_t	hplmn_scan_interval	Value in minutes. The duration after which device will start to look for HPLMN when camped on a roaming network.
uint8_t	ciot_pref_rat_scan_interval_valid	Must be set to TRUE if ciot_pref_rat_scan_interval is being passed.
uint32_t	ciot_pref_rat_scan_interval	Value in minutes. The duration after which device will start to look for a more preferred CIOT RAT when camped on a less preferred RAT in home.

8.1.2.31 struct qapi_Device_Info_Call_Req_t

QAPI device information voice call request structure.

Data fields

Type	Parameter	Description
voice_call_req_type	call_req_type	VoLTE call request.
union qapi_Device_Info_Call_Req_t	callReq	

8.1.2.32 union qapi_Device_Info_Call_Req_t.callReq

Data fields

Type	Parameter	Description
char	calling_number	VoLTE calling number.
callReq	endCallReq	
callReq	answerCallReq	

8.1.2.33 struct qapi_Device_Info_Call_Req_t.callReq.endCallReq

Data fields

Type	Parameter	Description
uint8_t	call_id	VoLTE call ID.
qapi_Device- _Info_Call_- Reject_Cause_t	reject_cause	Call end cause.

8.1.2.34 struct qapi_Device_Info_Call_Req_t.callReq.answerCallReq

Data fields

Type	Parameter	Description
uint8_t	call_id	VoLTE call ID.
bool	reject_call	Set to FALSE to answer call, TRUE to reject call.
qapi_Device- _Info_Call_- Reject_Cause_t	reject_cause	Call end cause.

8.1.2.35 struct qapi_Device_Info_Set_QCPDPP_Req_t

QAPI device information QCPDPP set request structure.

Type	Parameter	Description
qapi_Device- _Info_PDP_- Type_e	pdp_type	PDP type (mandatory). Range (0-4).
bool	apn_name_ valid	APN name field is set or not.
char	apn_name	APN name (mandatory).
bool	auth_type_valid	Authentication preference field is set or not.
qapi_Device- _Info_Auth_- Type_e	auth_type	Authentication preference. Range (0-3).
bool	username_valid	Username field is set or not.
char	username	Username.
bool	password_valid	Password field is set or not.
char	password	Password.

8.1.2.36 struct qapi_Device_Info_Set_QCPDPP_Rsp_t

QAPI device information QCPDPP set response structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Response_code	response	Response value.

8.1.2.37 struct qapi_Device_Info_Get_QCPDPP_Req_t

QAPI device information QCPDPP get request structure.

Data fields

Type	Parameter	Description
char	placeholder	

8.1.2.38 struct qapi_Device_Info_Get_QCPDPP_t

QAPI device information QCPDPP get structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Type_e	pdp_type	PDP type (mandatory).
bool	apn_name_- valid	APN name field is set or not.
char	apn_name	APN name (mandatory).
bool	auth_type_valid	Authentication preference field is set or not.
qapi_Device- _Info_Auth_- Type_e	auth_type	Authentication preference.
bool	username_valid	Username field is set or not.
char	username	Username.

8.1.2.39 struct qapi_Device_Info_Get_QCPDPP_Rsp_t

QAPI device information QCPDPP get response structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Response_code	response	Response value.
uint8_t	count	Number of profiles returned.
qapi_Device_- Info_Get_QCP- DPP_t *	list	List of profile settings.

8.1.2.40 struct qapi_Device_Info_Addr_t

QAPI device information IP address structure.

Data fields

Type	Parameter	Description
char	valid_addr	Indicates whether a valid address is available.
union qapi_- device_info_ip- _address_u	addr	Union of IP addresses.

8.1.2.41 union qapi_Device_Info_Addr_t::qapi_device_info_ip_address_u

Union of IP addresses.

Data fields

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

8.1.2.42 struct qapi_Device_Info_PDP_Address_t

QAPI device information PDP address information.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Type_e	pdp_type	PDP type (mandatory).
bool	apn_name_- valid	APN name field is set or not.
char	apn_name	APN name (mandatory).
qapi_Device_- Info_Addr_t	ipv4_addr	IPv4 address information.
qapi_Device_- Info_Addr_t	ipv6_addr	IPv6 address information.

8.1.2.43 struct qapi_Device_Info_Set_CGDCONT_Req_t

QAPI device information CGDCONT set request structure.

Type	Parameter	Description
qapi_Device- _Info_PDP_- Address_t	addr_info	PDP type, APN name, and IP address information. PDP IP address information not currently supported.
bool	d_comp_valid	Data compression field is set or not. Range (0-3).
qapi_Device- _Info_PDP_- Data_Compr_- Type_e	d_comp	Data compression type.
bool	h_comp_valid	Header compression field is set or not.
qapi_Device- _Info_PDP_Hdr- _Compr_Type- _e	h_comp	Header compression type. Range (0-4).
bool	ipv4AddrAlloc- _valid	IPv4AddrAlloc field is set or not.
uint8_t	ipv4AddrAlloc	Type of IPv4 address allocation (not supported).
bool	delete_profile_- valid	Delete profile field is set or not.
bool	delete_profile	Delete profile corresponding to PDP type and APN name passed.

8.1.2.44 struct qapi_Device_Info_Set_CGDCONT_Rsp_t

QAPI device information CGDCONT set response structure.

Data fields

Type	Parameter	Description
qapi_Device-Info_PDP_-Response_code	response	Response value.

8.1.2.45 struct qapi_Device_Info_Get_CGDCONT_Req_t

QAPI device information CGDCONT get request structure.

Data fields

Type	Parameter	Description
char	placeholder	

8.1.2.46 struct qapi_Device_Info_Get_CGDCONT_t

QAPI device information CGDCONT get structure.

Data fields

Type	Parameter	Description
qapi_Device-Info_PDP_-Address_t	addr_info	PDP type, APN name, and IP address information. PDP IP address information not currently supported.
bool	d_comp_valid	Data compression field is set or not.
qapi_Device-Info_PDP_-Data_Compr_-Type_e	d_comp	Data compression type.
bool	h_comp_valid	Header compression field is set or not.

qapi_Device_Info_PDP_Hdr_Compr_Type_e	h_comp	Header compression type.
bool	ipv4AddrAlloc_valid	IPv4AddrAlloc field is set or not.
uint8_t	ipv4AddrAlloc	Type of IPv4 address allocation. Always returns 0.

8.1.2.47 struct qapi_Device_Info_Get_CGDCONT_Rsp_t

QAPI device information CGDCONT get response structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_PDP_Response_code	response	Response value.
uint8_t	count	Number of profiles returned.
qapi_Device_Info_Get_CGDCONT_t *	list	List of profile settings.

8.1.2.48 struct qapi_Device_Info_Set_QCPDPIMSCFGE_Req_t

QAPI device information QCPDPIMSCFGE set request structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Type_e	pdp_type	PDP type (mandatory).
bool	apn_name_- valid	APN name field is set or not.
char	apn_name	APN name (mandatory).
bool	address_flag_- valid	Address flag field is set or not.
bool	address_flag	Address flag field (not supported).
bool	dhcp_flag_valid	DHCP flag field is set or not.
bool	dhcp_flag	PCSCF address using PCO Flag.
bool	cn_flag_valid	CN flag field is set or not.
bool	cn_flag	IM CN flag.

8.1.2.49 struct qapi_Device_Info_Set_QCPDPIMSCFGE_Rsp_t

QAPI device information QCPDPIMSCFGE set response structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Response_code	response	Response value.

8.1.2.50 struct qapi_Device_Info_Get_QCPDPIMSCFGE_Req_t

QAPI device information QCPDPIMSCFGE get request structure.

Data fields

Type	Parameter	Description
char	placeholder	

8.1.2.51 struct qapi_Device_Info_Get_QCPDPIMSCFGE_t

QAPI device info QCPDPIMSCFGE get structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Type_e	pdp_type	PDP type (mandatory).
bool	apn_name_- valid	APN name field is set or not.
char	apn_name	APN name (mandatory).
bool	address_flag_- valid	Address flag field is set or not.
bool	address_flag	Address flag field (not supported).
bool	dhcp_flag_valid	DHCP flag field is set or not.
bool	dhcp_flag	PCSCF address using PCO Flag.
bool	cn_flag_valid	CN flag field is set or not.
bool	cn_flag	IM CN flag.

8.1.2.52 struct qapi_Device_Info_Get_QCPDPIMSCFGE_Rsp_t

QAPI device information QCPDPIMSCFGE get response structure.

Data fields

Type	Parameter	Description
qapi_Device-Info_PDP_-Response_code	response	Response value.
uint8_t	count	Number of profiles returned.
qapi_Device-Info_Get_QCPDPCFGE_t	list	List of profile settings.
*		

8.1.2.53 struct qapi_Device_Info_MNC_t

QAPI device information MNC structure.

Data fields

Type	Parameter	Description
uint16_t	mnc	A 16-bit integer representation of MNC.
uint8_t	mnc_includes_pcs_digit	MNC PCS digit include status.

8.1.2.54 struct qapi_Device_Info_Set_QCPDPCFGE_Req_t

QAPI device information QCPDPCFGE set request structure.

Data fields

Type	Parameter	Description
qapi_Device-	pdp_type	PDP type (mandatory).

_Info_PDP_- Type_e		
bool	apn_name_- valid	APN name field is set or not.
char	apn_name	APN name (mandatory).
bool	apn_disable_- flag_valid	APN disable flag field is set or not.
bool	apn_disable_- flag	APN disabled flag.
bool	timer_value_- valid	Timer value field is set or not.
uint32_t	timer_value	PDN inactivity timeout (in seconds). Range (0-122820).
bool	apn_class_valid	APN class field is set or not.
uint8_t	apn_class	APN class. Range (0-16).
bool	apn_bearer_- valid	APN bearer field is set or not.
uint64_t	apn_bearer	APN bearer.
bool	max_pdn_- conn_per_- block_valid	Maximum PDN connections per block field is set or not.
uint16_t	max_pdn_- conn_per_block	Maximum PDN connections per time block. Range (0-1023).
bool	max_pdn_- conn_timer_- valid	Maximum PDN connections timer field is set or not.
uint16_t	max_pdn_- conn_timer	Maximum PDN connections timer (in secs). Range (0-3600).
bool	pdn_req_wait_- timer_valid	PDN request wait timer field is set or not.

uint16_t	pdn_req_wait_ timer	PDN request wait timer (in seconds). Range (0-1023).
bool	emergency_ calls_ supported_valid	Emergency call supported field is set or not.
bool	emergency_ calls_supported	Emergency call (not supported).
bool	operator_ reserved_pco_ valid	Operator reserved PCO field is set or not.
uint16_t	operator_ reserved_pco	Operator reserved PCO ID. Range (65280-65535).
bool	mcc_valid	MCC field is set or not.

Type	Parameter	Description
uint16_t	mcc	Mobile Country Code. Range (0-999).
bool	mnc_valid	MNC field is set or not.
qapi_Device_ Info_MNC_t	mnc	Mobile Network Code. Range (0-999).

8.1.2.55 struct qapi_Device_Info_Get_QCPDPCFGE_Req_t

QAPI device information QCPDPCFGE get request structure.

Data fields

Type	Parameter	Description
char	placeholder	

8.1.2.56 struct qapi_Device_Info_Set_QCPDPCFGE_Rsp_t

QAPI device information QCPDPCFGE set response structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Response_code	response	Response value.

8.1.2.57 struct qapi_Device_Info_Get_QCPDPCFGE_t

QAPI device information QCPDPCFGE get structure.

Data fields

Type	Parameter	Description
qapi_Device- _Info_PDP_- Type_e	pdp_type	PDP type (mandatory).
bool	apn_name_- valid	APN name field is set or not.
char	apn_name	APN name (mandatory).
bool	apn_disable_- flag_valid	APN disable flag field is set or not.
bool	apn_disable_- flag	APN disabled flag.
bool	timer_value_-	Timer value field is set or not.

	valid	
uint32_t	timer_value	PDN inactivity timeout.
bool	apn_class_valid	APN class field is set or not.
uint8_t	apn_class	APN class.
bool	apn_bearer_valid	APN bearer field is set or not.
uint64_t	apn_bearer	APN bearer.
bool	max_pdn_conn_per_block_valid	Maximum PDN connections per block field is set or not.
uint16_t	max_pdn_conn_per_block	Maximum PDN connections per time block.
bool	max_pdn_conn_timer_valid	Maximum PDN connections timer field is set or not.
uint16_t	max_pdn_conn_timer	Maximum PDN connections timer.
bool	pdn_req_wait_timer_valid	PDN request wait timer field is set or not.
uint16_t	pdn_req_wait_timer	PDN request wait timer.
bool	emergency_calls_supported_valid	Emergency call supported field is set or not.
bool	emergency_calls_supported	Emergency call (not supported).
bool	operator_reserved_pco_valid	Operator reserved PCO field is set or not.

uint16_t	operator_ - reserved_pco	Operator reserved PCO ID.
bool	mcc_valid	MCC field is set or not.

FIBOCOM
Confidential

Type	Parameter	Description
uint16_t	mcc	Mobile Country Code.
bool	mnc_valid	MNC field is set or not.
qapi_Device_Info_MNC_t	mnc	Mobile Network Code.

8.1.2.58 struct qapi_Device_Info_Get_QCPDPCFGE_Rsp_t

QAPI device information QCPDPCFGE get response structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_PDP_Response_code	response	Response value.
uint8_t	count	Number of profiles returned.
qapi_Device_Info_Get_QCPDPCFGE_t *	list	List of profile settings.

8.1.2.59 struct qapi_Device_Info_FSK_Start_Req_t

QAPI device information FSK start request structure.

Data fields

Type	Parameter	Description
uint8_t	baudrate_valid	Must be set to TRUE if baudrate is being passed.

uint8_t	baudrate	<p>Baudrate at which FSK transmission will start.</p> <p>Possible values:</p> <p>0 – 3466bps</p> <p>1 – 1733bps</p> <p>2 – 866bps</p> <p>If TLV is not present, the default value available at GL1 will be used.</p>
uint8_t	gaptime_valid	Must be set to TRUE if gaptime is being passed.
uint16_t	gaptime	<p>Possible values depend on baudrate value, with respect to the number of bits per TDMA frame. Maximum allowed bits: 1024</p> <p>If TLV is not present, the default value available at GL1 will be used.</p>
uint8_t	restart_mode_valid	Must be set to TRUE if restart_mode is being passed.
uint8_t	restart_mode	<p>0 – Sends data using all channels/frequencies defined with QMI_NAS_SET_FSK_HOP_TABLE_REQ_MSG (default)</p> <p>– Shall work as 0 but indicates storing the next frequency index of the hopping table when FSKSTOP command is sent.</p>

8.1.2.60 struct qapi_Device_Info_FSK_Data_Req_t

QAPI device information FSK data request structure.

Data fields

Type	Parameter	Description
uint32_t	data_len	Must be set to number of elements in data.
uint8_t	data	FSK Data to be sent to GERAN.

8.1.2.61 struct qapi_Device_Info_Get_FSK_Debug_Rsp_t

QAPI device information get FSK debug parameters response structure.

Data fields

Type	Parameter	Description
uint8_t	delta_valid	Must be set to TRUE if delta is being passed.
uint8_t	delta	Delta of frequency (in kHz). Possible values: 1, 2, 3, 5. If TLV is not present, the default available at GL1 will be used.
uint8_t	freq_valid	Must be set to TRUE if frequency is being passed.
uint32_t	freq	Frequency for debugging. Possible values: 902000-928000, 966037-967957.
uint8_t	baudrate_valid	Must be set to TRUE if baudrate is being passed.
uint8_t	baudrate	Baudrate at which FSK transmission will start. Possible values: 0 – 3466bps 1 – 1733bps 2 – 866bps

		If TLV is not present, the default value available at GL1 will be used.
--	--	---

FIBOCOM
Confidential

8.1.2.62 struct qapi_Device_Info_Set_FSK_Debug_Req_t

QAPI device information set FSK debug parameters request structure.

Data fields

Type	Parameter	Description
uint32_t	freq	Frequency for debugging. Possible values: 902000-928000, 966037-967957.
uint8_t	delta_valid	Must be set to TRUE if delta is being passed.
uint8_t	delta	Delta of frequency (in kHz). Possible values: 1, 2, 3, 5. If TLV is not present, the default value available at GL1 will be used.
uint8_t	baudrate_valid	Must be set to TRUE if baudrate is being passed.
uint8_t	baudrate	Baudrate at which FSK transmission will start. Possible values: 0 – 3466bps 1 – 1733bps 2 – 866bps If TLV is not present, the default value available at GL1 will be used.

8.1.2.63 struct qapi_Device_Info_FSK_Hoptable_Entry_Type

QAPI device information FSK hop table entry structure.

Data fields

Type	Parameter	Description
uint8_t	delta	Delta of frequency (in kHz). Possible values: 1, 2, 3, 5.
uint32_t	freq_len	Must be set to number of elements in frequency.
uint32_t	freq	Possible values: 902000-928000, 966037-967957. Frequencies through which the module can automatically hop (in KHz).

8.1.2.64 struct qapi_Device_Info_Set_FSK_Hoptable_Req_t

QAPI device information set hop table contents per index request structure.

Data fields

Type	Parameter	Description
uint8_t	index	Index of record in hop table that must be set. Possible values: 0, 1, 2.
qapi_Device- _Info_FSK- _Hoptable_- Entry_Type	fsk_hop_table	

8.1.2.65 struct qapi_Device_Info_Get_FSK_Hoptable_Rsp_t

QAPI device information get hop table contents response structure.

Data fields

Type	Parameter	Description
uint8_t	fsk_hop_table- _valid	Must be set to TRUE if fsk_hop_table is being passed.
uint32_t	fsk_hop_table- _len	Must be set to number of elements in fsk_hop_table.
qapi_Device- _Info_FSK- _Hoptable_- Entry_Type	fsk_hop_table	

8.1.2.66 struct qapi_Device_Info_Get_FSK_PCL_Rsp_t

QAPI device information get PCL response structure.

Data fields

Type	Parameter	Description
uint8_t	pcl_gsm_valid	Must be set to TRUE if pcl_gsm is being passed.
uint8_t	pcl_gsm	PCL for GSM band. Range: (5-19, 99) Default: 5 If TLV is not present, the default value available at GL1 will be used.
uint8_t	pcl_dcs_valid	Must be set to TRUE if pcl_dcs is being passed.
uint8_t	pcl_dcs	PCL for DCS band. Range: (0-15, 99) Default: 99. If TLV is not present, the default value available at GL1 will be used.
uint8_t	pcl_pcs_valid	Must be set to TRUE if pcl_pcs is being passed.
uint8_t	pcl_pcs	PCL for PCS band. Range: (0-15, 99) Default: 99. If TLV is not present, the default value available at GL1 will be used.

8.1.2.67 struct qapi_Device_Info_Set_FSK_PCL_Req_t

QAPI device info set PCL request structure.

Data fields

Type	Parameter	Description
uint8_t	pcl_gsm_valid	Must be set to TRUE if pcl_gsm is being passed
uint8_t	pcl_gsm	PCL for GSM band. Range: (5-19, 99) Default: 5. If TLV is not present, the default value available at GL1 will be used.
uint8_t	pcl_dcs_valid	Must be set to TRUE if pcl_dcs is being passed
uint8_t	pcl_dcs	PCL for DCS band. Range: (0-15, 99) Default: 99 If TLV is not present, the default value available at GL1 will be used.
uint8_t	pcl_pcs_valid	Must be set to true if pcl_pcs is being passed
uint8_t	pcl_pcs	PCL for PCS band. Range: (0-15, 99) Default: 99 If TLV is not present, the default value available at GL1 will be used.

8.1.2.68 struct qapi_Device_Info_Request_t

QAPI device information request structure.

Data fields

Type	Parameter	Description
union qapi_Device_Info_Request_t	req	

8.1.2.69 union qapi_Device_Info_Request_t.req

Data fields

Type	Parameter	Description
qapi_Device- _Info_Req_- Verify_Pin_t *	pin_verify_req	PIN verify request data.
qapi_Device- _Info_Req_- Change_Pin_t *	pin_change_req	PIN change request data.
qapi_Device- _Info_Req_- Unblock_Pin_t *	pin_unblock_ req	PIN unblock request data .
qapi_Device- _Info_Req_- Protect_Pin_t *	pin_protect_req	PIN protect request data.
qapi_Device_- Info_Req_Set_- FDN_t *	set_fdn_req	Sets FDN request data.
qapi_Device_- Info_APDU_- Command_t *	apdu_cmd_req	Sends APDU command request data.
qapi_Device_- Info_PREFERRED- plmn_set_req	preferred_- plmn_set_req	Sets preferred PLMN items request data.

_PLMN_Set_- Req_t *		
qapi_Device_- Info_Preferred- _PLMN_Read- _Req_t *	preferred_- plmn_read_req	Reads preferred PLMN items request data.
qapi_Device_- Info_Req_ED- RX_Set_t *	set_edrx_req	EDRX sets request data.
qapi_Device_- Info_Req_ED- RX_Get_t *	get_edrx_req	EDRX gets request data.
qapi_Device_- Info_Jamming- _Status_Get_- Req_t *	jamming_- status_req	Gets jamming status request data.
qapi_Device_- Info_Call_Req- _t *	call_req	VoLTE call request data.

Type	Parameter	Description
qapi_Device_ - Info_Set_QCP- DPP_Req_t *	set_qcpdpp_req	QCPDPP command set request data.
qapi_Device_ - Info_Get_QCP- DPP_Req_t *	get_qcpdpp_req	QCPDPP command get request data.
qapi_Device_ - Info_Set_CG- DCONT_Req_t *	set_cgdcont_ - req	CGDCONT command set request data.
qapi_Device_ - Info_Get_CG- DCONT_Req_t *	get_cgdcont_ - req	CGDCONT command get request data.
qapi_Device_ - Info_Set_QCP- DPIMSCFGE_- Req_t *	set_- qcpdpimscfg_- req	QCPDPIMSCFGE command set request data.
qapi_Device_ - Info_Get_QCP- DPIMSCFGE_- Req_t *	get_- qcpdpimscfg_- req	QCPDPIMSCFGE command get request data.
qapi_Device_ - Info_Set_QCP- DPCFGE_Req- _t *	set_qcpdpcfg_- req	QCPDPCFGE command set request data.

qapi_Device_- Info_Get_QCP- DPCFGE_Req- _t *	get_qcpdpcfg_- req	QCPDPCFGE command get request data.
qapi_Device- _Info_FSK- _Start_Req_t *	fsk_start_req	FSK start request data.
qapi_Device- _Info_FSK- _Data_Req_t *	fsk_data_req	FSK data request data.
qapi_Device_- Info_Set_FSK- _Debug_Req_t *	set_fsk_debug- _req	FSK set debug parameters request data.
qapi_Device_- Info_Set_FS- K_Hoptable_- Req_t *	set_fsk_- hoptable_req	FSK set hop table parameters request data.

Type	Parameter	Description
qapi_Device_Info_Set_FSK_PCL_Req_t *	set_fsk_pcl_req	FSK set pcl parameters request data.

8.1.2.70 struct qapi_Device_Info_Response_t

QAPI device information response structure.

Data fields

Type	Parameter	Description
union qapi_Device_Info_Response_t	resp	

8.1.2.71 union qapi_Device_Info_Response_t.resp

Data fields

Type	Parameter	Description
qapi_Device- _Info_Pin_- Response_t *	pin_resp	PIN operation response data.
qapi_Device_- Info_Get_FDN- _Response_t *	get_fdn_rsp	Gets FDN response data.
qapi_Device_- Info_APDU_- Response_t *	apdu_cmd_rsp	Sends APDU command response data.
qapi_Device- _Info_UIM- _Response_t *	preferred_- plmn_set_rsp	Sets preferred PLMN response data.
qapi_Device_- Info_PREFERRED- _PLMN_Read- _Rsp_t *	preferred_- plmn_read_rsp	Reads preferred PLMN items response data.
qapi_Device_- Info_UICC_- Application_- Rsp_t *	uicc_apps_- read_rsp	Reads UICC application response data.
qapi_Device_- Info_Rsp_ED- RX_Get_t *	get_edrx_rsp	EDRX get response data.
qapi_Device_- Info_Jamming- _Status_Get_- Rsp_t *	jamming_- status_rsp	Gets jamming status response data.

qapi_Device- _Info_IMS_- Registration_- Status_Rsp_t *	ims_- registration_rsp	Gets IMS registration status.
qapi_Device- _Info_Call_- Status_Rsp_t *	call_status_rsp	Gets VoLTE call status
qapi_Device_- Info_Set_QCP- DPP_Rsp_t *	set_qcpdpp_rsp	QCPDPP command set response data.
qapi_Device_- Info_Get_QCP- DPP_Rsp_t *	get_qcpdpp_rsp	QCPDPP command get response data.

Type	Parameter	Description
qapi_Device_ - Info_Set_CG- DCONT_Rsp_t *	set_cgdcont_ - rsp	CGDCONT command set response data.
qapi_Device_ - Info_Get_CG- DCONT_Rsp_t *	get_cgdcont_ - rsp	CGDCONT command get response data.
qapi_Device_ - Info_Set_QCP- DPIMSCFGE_ - Rsp_t *	set_ - qcpdpimscfg_ - rsp	QCPDPIMSCFGE command set response data.
qapi_Device_ - Info_Get_QCP- DPIMSCFGE_ - Rsp_t *	get_ - qcpdpimscfg_ - rsp	QCPDPIMSCFGE command get response data.
qapi_Device_ - Info_Set_QCP- DPCFGE_Rsp- _t *	set_qcpdpcfg_ - rsp	QCPDPCFGE command set response data.
qapi_Device_ - Info_Get_QCP- DPCFGE_Rsp- _t *	get_qcpdpcfg_ - rsp	QCPDPCFGE command get response data.
qapi_Device_ - Info_Scan_ - Config_Rsp_t *	scan_config_ - rsp	Gets scan configurations.

qapi_Device_Info_Get_FSK_Debug_Rsp_t *	get_fsk_debug_rsp	FSK get debug parameters response data.
qapi_Device_Info_Get_FSK_Hoptable_Rsp_t *	get_fsk_hoptable_rsp	FSK get hop table parameters response data.
qapi_Device_Info_Get_FSK_PCL_Rsp_t *	get_fsk_pcl_rsp	FSK get pcl parameters response data.

8.1.2.72 struct qapi_Device_Info_PLMN_Info_t

QAPI PLMN scan result information structure.

Data fields

Type	Parameter	Description
uint16_t	mcc	A 16-bit integer representation of MCC. Range: 0 to 999.
uint16_t	mnc	A 16-bit integer representation of MNC. Range: 0 to 999.
qapi_Device_Info_PLMN_Act_Type_t	Act	0 - GSM, 7 - LTE_M1, 9 - LTE_NB1.

8.1.2.73 struct qapi_Device_Info_PLMN_Info_List_t

QAPI PLMN information list structure.

Data fields

Type	Parameter	Description
uint32_t	list_len	Set to number of elements in PLMN_list.
qapi_Device_Info_PLMN_Info_t	plmn_list	PLMN scan result, plmn_list.

8.1.3 Data Typedef

8.1.3.1 typedef void *qapi_Device_Info_Hndl_t

Device information handle.

8.1.3.2 typedef void (*qapi_Device_Info_Callback_t)(const qapi_Device_Info_t *info)

QAPI device information callback typedef.

Deprecated in DAM space. Use callback qapi_Device_Info_Callback_v2() instead.

8.1.3.3 typedef void (*qapi_Device_Info_Callback_t_v2)(qapi_Device_Info_Hndl_t device_info_hndl, const qapi_Device_Info_t *info)

QAPI device information callback_v2 typedef.

8.1.4 Data Enumeration

8.1.4.1 enum acq_order_pref_enum

Valid values for Ciot_Acquisition_order ([QAPI_DEVICE_INFO_RAT_ACQ_PREF_E](#)).

Enumerator:

QAPI_DEVICE_INFO_ACQ_ORDER_PREF_NO_SVC	No Service.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_AMPS	AMPS.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_GSM	GSM.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_UMTS	UMTS.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_WLAN	WLAN.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_GPS	GPS.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_LTE	LTE.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_LTE_M1	LTE M1.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_LTE_NB1	LTE NB1.
QAPI_DEVICE_INFO_ACQ_ORDER_PREF_NO_CHANGE	No change.

8.1.4.2 enum battery_status

Valid values for battery status ([QAPI_DEVICE_INFO_BATTERY_STATUS_E](#)).

Enumerator:

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.		

8.1.4.3 enum srv_status

Valid values for network service status ([QAPI_DEVICE_INFO_SERVICE_STATE_E](#)).

Enumerator:



QAPI_DEVICE_INFO_SRV_STATE_NO_SRV No service.

QAPI_DEVICE_INFO_SRV_STATE_SRV Service is available.

8.1.4.4 enum nw_indication

Valid values for network indication ([QAPI_DEVICE_INFO_NETWORK_IND_E](#)).

Enumerator:

QAPI_DEVICE_INFO_NW_IND_NO_SRV No service.

QAPI_DEVICE_INFO_NW_IND_SRV Service is available.

8.1.4.5 enum rrc_state Valid values for network RRC state. **Enumerator:**

QAPI_DEVICE_INFO_RRC_IDLE Status: Idle.

QAPI_DEVICE_INFO_RRC_CONNECTED Status: connected.

8.1.4.6 enum emm_state

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

Enumerator:

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.

8.1.4.7 enum roaming_info

Valid values for network roaming status ([QAPI_DEVICE_INFO_ROAMING_E](#)).



Enumerator:

QAPI_DEVICE_INFO_ROAMING_STATUS_OFF Roaming status: OFF.

QAPI_DEVICE_INFO_ROAMING_STATUS_ON Roaming status: ON.

8.1.4.8 enum sim_state

Valid value for sim state ([QAPI_DEVICE_INFO_SIM_STATE_E](#)).

Enumerator:

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.

8.1.4.9 enum ciot_lte_op_mode

Enumerator:

QAPI_DEVICE_INFO_NO_SRV_V01 No service. **QAPI_DEVICE_INFO_LTE_WB_V01** Camped on LTE wideband. **QAPI_DEVICE_INFO_LTE_M1_V01** Camped on LTE M1.
QAPI_DEVICE_INFO_LTE_NB1_V01 Camped on LTE NB1.

8.1.4.10 enum client_group

Enumerator:

QAPI_DEVICE_INFO_GNSS_PRIORITY Positioning procedure priority is higher than usual communication and page reception.
QAPI_DEVICE_INFO_WWAN_PRIORITY Communication priority (including page reception) is higher than positioning procedures.

8.1.4.11 enum loaded_app_info

Enumerator:

QAPI_DEVICE_INFO_NONE None loaded/pending. **QAPI_DEVICE_INFO_WWAN_PENDING** WWAN pending state. **QAPI_DEVICE_INFO_GPS_PENDING** GPS pending state.
QAPI_DEVICE_INFO_WWAN_LOADED WWAN loaded state.
QAPI_DEVICE_INFO_GPS_LOADED GPS loaded state.

8.1.4.12 enum jammer_status

Enumerator:

QAPI_DEVICE_INFO_JAMMING_NOTFOUND Jamming status not found.
QAPI_DEVICE_INFO_JAMMING_FOUND Jamming status found.
QAPI_DEVICE_INFO_JAMMING_UNKNOWN Jamming status unknown.

8.1.4.13 enum voice_call_req_type

Enumerator:

QAPI_DEVICE_INFO_VOICE_DIAL_CALL_REQ Dial VoLTE call request.
QAPI_DEVICE_INFO_VOICE_END_CALL_REQ End VoLTE call request.
QAPI_DEVICE_INFO_VOICE_ANSWER_CALL_REQ Answer or reject VoLTE call request.

8.1.4.14 enum qapi_Device_Info_ID_t

Device information types.

Response type: Return value of the Device ID as specified in the enum qapi_Device_Info_Type_t.

Preparation:

Query - Queries the device information using the API [qapi_Device_Info_Get_v2\(\)](#).

Set - Sets the device information using the API [qapi_Device_Info_Set\(\)](#).

Indication - Sets the device information callback, registering for various indications using the API [qapi_Device_Info_Set_Callback_v2\(\)](#).

Enumerator:

QAPI_DEVICE_INFO_BUILD_ID_E Device BUILD_ID.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_IMEI_E Device IMEI.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_IMSI_E UIM IMSI.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_OS_VERSION_E Device OS version.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_MANUFACTURER_E Device manufacturer.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_MODEL_ID_E Device model ID.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_BATTERY_STATUS_E Device battery status.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_BATTERY_PERCENTAGE_E Device battery percentage.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_TIME_ZONE_E Device time zone.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_ICCID_E Device ICCID.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_4G_SIG_STRENGTH_E Network signal strength.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_BASE_STATION_ID_E Network base station ID.



Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_MCC_E Network MCC.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_MNC_E Network MNC.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_SERVICE_STATE_E Network service status.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_MDN_E Device MDN.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_TAC_E Network tracking area code.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_CELL_ID_E Network cell ID.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_RCCS_E Network RRC state.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_EMMS_E Network EMM state.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_SERVING_PCI_E Network serving cell PCI.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_SERVING_RSRQ_E Serving cell RSRQ.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_SERVING_EARFCN_E Serving cell EARFCN.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_NETWORK_IND_E Network indication.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_ROAMING_E Roaming status.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_LAST_POWER_ON_E Last power on time.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_CHIPID_STRING_E Chipset name.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query.

QAPI_DEVICE_INFO_SIM_STATE_E SIM state.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_NETWORK_BEARER_E Network bearer.



Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_LINK_QUALITY_E Network link quality.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Query and Indication.

QAPI_DEVICE_INFO_TX_BYTES_E Device Tx bytes.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Indication.

QAPI_DEVICE_INFO_RX_BYTES_E Device Rx bytes.

Response type: QAPI_DEVICE_INFO_TYPE_BUFFER_E. Operation: Indication.

QAPI_DEVICE_INFO_ANY Any device information.

QAPI_DEVICE_INFO_CIoT_LTE_OP_MODE_PREF_E Device LTE operational mode preference.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E. Operation: Set and Query.

QAPI_DEVICE_INFO_LTE_M1_BAND_PREF_E Device LTE M1 band preference.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E. Operation: Set and Query.

QAPI_DEVICE_INFO_LTE_NB1_BAND_PREF_E Device LTE NB1 band preference.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E. Operation: Set and Query.

QAPI_DEVICE_INFO_MODE_PREF_E Device mode preference.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E. Operation: Set and Query..

QAPI_DEVICE_INFO_RAT_ACQ_PREF_E RAT Acquisition order preference.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E. Operation: Set and Query.

QAPI_DEVICE_INFO_PS_DETACH_E Device PS Detach.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E. Operation: Set.

QAPI_DEVICE_INFO_PS_ATTACH_E Device PS Attach.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E. Operation: Set.

QAPI_DEVICE_INFO_PSM_TIMER_E Device PSM timer.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Set and Query.

QAPI_DEVICE_INFO_ACTIVE_TIMER_E Device power save mode active timer value.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Set and Query.

QAPI_DEVICE_INFO_LTE_OP_MODE_E Device LTE operational mode.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_LAC_E Location Area Code.



Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_RAT_TYPE_GSM_CELL_E Device Cell info RAT type - GSM. Response type:

QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_RAT_TYPE_LTE_M1_CELL_E Device Cell info RAT type - LTE M1.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_RAT_TYPE_LTE_NB1_CELL_E Device Cell info RAT type -LTE NB1 Cell info.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_BAND_GSM_CELL_E Device Cell info Band for RAT type GSM. Response type:

QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Set.

QAPI_DEVICE_INFO_BAND_LTE_M1_CELL_E Device Cell info Band for RAT type LTE M1.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Set.

QAPI_DEVICE_INFO_BAND_LTE_NB1_CELL_E Device Cell info Band for RAT type LTE NB1.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Set.

QAPI_DEVICE_INFO_SET_APP_PRIORITY_E Device App priority.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Set.

QAPI_DEVICE_INFO_GET_APP_PRIORITY_E Device App priority.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_GET_LOADED_TECH_E Device currently loaded technology. Response type:

QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Query.

QAPI_DEVICE_INFO_OPERATING_MODE_E Device operating mode.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E, 0 = Online mode, 1 = Low Power mode.

Operation: Set and Query.

QAPI_DEVICE_INFO_EDRX_E eDRX information.

Response type: QAPI_DEVICE_INFO_TYPE_EDRX_INFO_E. Operation: Indication.

QAPI_DEVICE_INFO_PLMN_LIST_E Scan PLMN list.

Response type: QAPI_DEVICE_INFO_TYPE_PLMN_LIST_E. Operation: Query.

QAPI_DEVICE_INFO_PLMN_SELECT_E Select PLMN.

Response type: QAPI_DEVICE_INFO_TYPE_ARRAY_E, Val[0]: Net Select Prefer, 0 = Auto, 1 = Manual;

Val[1]: MCC; Val[2]: MNC. Operation: Set.

QAPI_DEVICE_INFO_CEMODE_E CEMODE info.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E, 0 = PS mode2, 1 = CS/PS mode1, 2 =

CS/PS mode2. Operation: Set and Query.

QAPI_DEVICE_INFO_SHUTDOWN_E Shutdown UE.

Response type: Not applicable. Operation: Set.

QAPI_DEVICE_INFO_WMS_EVENT_REPORT_E WMS.

Response type: QAPI_DEVICE_INFO_TYPE_INTEGER_E. Operation: Indication.

QAPI_DEVICE_INFO_JAMMER_IND_E Jamming status.



Response type: [qapi_Device_Info_Jamming_Status_Get_Rsp_t](#). Operation: Indication.

QAPI_DEVICE_INFO_IMS_IOT_REGISTRATION_STATUS_IND_E Registration status.

Response type: [qapi_Device_Info_IMS_Registration_Status_Rsp_t](#). Operation: Indication.

QAPI_DEVICE_INFO_VOICE_ALL_CALL_STATUS_IND_E All call status.

Response type: [qapi_Device_Info_Call_Status_Rsp_t](#). Operation: Indication.

QAPI_DEVICE_INFO_SUBS_ENABLE_E Enable/disable subscription.

Response type: Not applicable. Operation: Set.

QAPI_DEVICE_INFO_MODEM_SERVICE_STATE_E Modem service state: Loaded or not loaded.

Response type: [QAPI_DEVICE_INFO_TYPE_BOOLEAN_E](#) Operation: Indication and Query.

QAPI_DEVICE_INFO_WWAN_SLEEP_INFO_E WWAN sleep duration information.

Response type: [QAPI_DEVICE_INFO_TYPE_INTEGER_E](#) Operation: Indication.

QAPI_DEVICE_INFO_WWAN_SLEEP_THRESHOLD_E WWAN sleep threshold.

Response type: [QAPI_DEVICE_INFO_TYPE_INTEGER_E](#) Operation: Set and Query

8.1.4.15 enum qapi_Device_Info_Req_ID_t

Device information request types.

Enumerator:

QAPI_DEVICE_INFO_REQ_VERIFY_PIN_E Verify PIN. **QAPI_DEVICE_INFO_REQ_CHANGE_PIN_E**

Change PIN. **QAPI_DEVICE_INFO_REQ_UNBLOCK_PIN_E** Unblock PIN.

QAPI_DEVICE_INFO_REQ_PROTECT_PIN_E Enable/disable PIN.

QAPI_DEVICE_INFO_REQ_SET_FDN_STATUS_E Enable/disable FDN Service.

QAPI_DEVICE_INFO_REQ_GET_FDN_STATUS_E Enable/disable FDN Service.

QAPI_DEVICE_INFO_REQ_SEND_APDU_E Send APDU.

QAPI_DEVICE_INFO_REQ_READ_PREFERRED_PLMN_E Read preferred PLMN list.

QAPI_DEVICE_INFO_REQ_WRITE_PREFERRED_PLMN_E Write preferred PLMN item.

QAPI_DEVICE_INFO_REQ_READ_UICC_APPLICATIONS_E Read UICC Application list.

QAPI_DEVICE_INFO_REQ_EDRX_SET_E Set edrx parameters.

QAPI_DEVICE_INFO_REQ_EDRX_GET_E Get edrx parameters.

QAPI_DEVICE_INFO_REQ_GET_JAMMER_STATUS_E Get jamming status.

QAPI_DEVICE_INFO_REQ_GET_IMS_REGISTRATION_E Get IMS registration status.

QAPI_DEVICE_INFO_REQ_VOLTE_CALL_E Dial, answer, or reject VoLTE call.

QAPI_DEVICE_INFO_REQ_SET_QCPDPP_E Set profile authentication parameters.

QAPI_DEVICE_INFO_REQ_SET_CGDCONT_E Set basic PDP context parameters.



QAPI_DEVICE_INFO_REQ_SET_QCPDPIMSCFG_E Set IMS related PDP profile parameters.

QAPI_DEVICE_INFO_REQ_SET_QCPDPCFG_E Set APN related PDP profile parameters.

QAPI_DEVICE_INFO_REQ_GET_QCPDPP_E Fetch profile authentication parameters.

QAPI_DEVICE_INFO_REQ_GET_CGDCONT_E Fetch basic PDP context parameters.

QAPI_DEVICE_INFO_REQ_GET_QCPDPIMSCFG_E Fetch IMS related PDP profile parameters.

QAPI_DEVICE_INFO_REQ_GET_QCPDPCFG_E Fetch APN related PDP profile parameters.

QAPI_DEVICE_INFO_SCAN_CONFIG_E Get Scan Counter Configuration parameters. >

QAPI_DEVICE_INFO_REQ_FSK_START_E FSK start request.

QAPI_DEVICE_INFO_REQ_FSK_STOP_E FSK stop request.

QAPI_DEVICE_INFO_REQ_FSK_DATA_E FSK data request.

QAPI_DEVICE_INFO_REQ_GET_FSK_DEBUG_E Get FSK debug parameters request.

QAPI_DEVICE_INFO_REQ_SET_FSK_DEBUG_E Set FSK debug parameters request.

QAPI_DEVICE_INFO_REQ_GET_FSK_HOP_TABLE_E Get FSK hop table parameters request.

QAPI_DEVICE_INFO_REQ_SET_FSK_HOP_TABLE_E Set FSK hop table parameters request.

QAPI_DEVICE_INFO_REQ_GET_FSK_PCL_E Get FSK pcl parameters request.

QAPI_DEVICE_INFO_REQ_SET_FSK_PCL_E Set FSK pcl parameters request.

8.1.4.16 enum qapi_Device_Info_Type_t

Device information response types.

Enumerator:

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.

QAPI_DEVICE_INFO_TYPE_ARRAY_E Response type is array.

QAPI_DEVICE_INFO_TYPE_CELL_INFO_E Response type is qapi_cell_info type.

QAPI_DEVICE_INFO_TYPE_EDRX_INFO_E Response type is [qapi_Device_Info_Rsp_EDRX_Get_t](#).

QAPI_DEVICE_INFO_TYPE_PLMN_LIST_E Response type is [qapi_Device_Info_PLMN_Info_List_t](#).

QAPI_DEVICE_INFO_TYPE_JAMMING_STATUS_E Response type is [qapi_Device_Info_Jamming_Status_Get_Rsp_t](#).

QAPI_DEVICE_INFO_TYPE_IMS_REGISTRATION_STATUS_E Response type is [qapi_Device_Info_IMS_Registration_Status_Rsp_t](#).

QAPI_DEVICE_INFO_TYPE_VOLTE_CALL_STATUS_E Response type is [qapi_Device_Info_Call_Status_Rsp_t](#).

8.1.4.17 enum qapi_Device_Info_Pin_Id_t

Device information PIN ID types.

Enumerator:

QAPI_DEVICE_INFO_PIN_ID_PIN1 Pin1.

QAPI_DEVICE_INFO_PIN_ID_PIN2 Pin2.

QAPI_DEVICE_INFO_PIN_ID_UNIVERSER_PIN Universer pin.

8.1.4.18 enum qapi_Device_Info_UIM_Response_code

Device information UIM error types.

Enumerator:



QAPI_DEVICE_INFO_UIM_NO_ERROR	No	UIM	error	return.
QAPI_DEVICE_INFO_UIM_INCORRECT_PIN	Incorrent		PIN	input.
QAPI_DEVICE_INFO_UIM_PIN_BLOCKED	PIN is blocked.			
QAPI_DEVICE_INFO_UIM_PIN_PERM_BLOCKED	PIN	is	blocked	permanently.
QAPI_DEVICE_INFO_UIM_ACCESS_DENIED	Access	denied	from	UIM.
QAPI_DEVICE_INFO_UIM_INTERNAL_ERROR	Internal error.			

FIBOCOM

Confidential

8.1.4.19 enum qapi_Device_Info_FDN_Status_t

Device information FDN status types.

Enumerator:

QAPI_DEVICE_INFO_FDN_NOT_AVAILABLE FDN is not available.

QAPI_DEVICE_INFO_FDN_AVAILABLE_DISABLED FDN is available but disabled.

QAPI_DEVICE_INFO_FDN_AVAILABLE_ENABLED FDN is available and enabled.

8.1.4.20 enum qapi_Device_Info_PREFERRED_PLMN_List_ID_t

Device information preferred PLMN List type types.

Enumerator:

QAPI_DEVICE_INFO_PREFERRED_PLMN_LIST_OPLMNWACT PLMN list from EFplmnwact (User controlled PLMN selector with Access Technology).

QAPI_DEVICE_INFO_PREFERRED_PLMN_LIST_HPLMNWACT PLMN list from EFoplmnwact (Operator controlled PLMN selector with Access Technology).

8.1.4.21 enum qapi_Device_Info_EDRX_Act_Type_t

QAPI eDRX AcT type define.

Enumerator:

QAPI_DEVICE_INFO_EDRX_ACT_NOT_USED AcT is not specified.

QAPI_DEVICE_INFO_EDRX_ACT_LTE_M1 LTE Cat M1.

QAPI_DEVICE_INFO_EDRX_ACT_LTE_NB1 LTE Cat NB1.

8.1.4.22 enum qapi_Device_Info_RAT_Type_t

Device information RAT types.

Enumerator:

QAPI_DEVICE_INFO_LTE_NB1 LTE NB1.

8.1.4.23 enum qapi_Device_Info_IMS_Iot_Reg_Status_t

ims_iot registration status.

Enumerator:

QAPI_DEVICE_INFO_IMS_IOT_STATUS_MIN_V01 Invalid registration status for IMS.

QAPI_DEVICE_INFO_IMS_IOT_STATUS_NOT_REGISTERED_V01 Not registered or registration failed state for IMS.

QAPI_DEVICE_INFO_IMS_IOT_STATUS_REGISTERING_V01 Registering for IMS.

QAPI_DEVICE_INFO_IMS_IOT_STATUS_REGISTERED_V01 Registered for IMS.

QAPI_DEVICE_INFO_IMS_IOT_STATUS_LIMITED_REGISTERED_V01 Limited registration for IMS.

QAPI_DEVICE_INFO_IMS_IOT_STATUS_MAX_V01 Invalid registration status for IMS.

8.1.4.24 enum qapi_Device_Info_Call_Reject_Cause_t

Voice call reject cause.

Enumerator:

QAPI_DEVICE_INFO_Call_REJECT_CAUSE_INVALID Invalid value.

QAPI_DEVICE_INFO_Call_REJECT_CAUSE_USER_BUSY User is busy.

QAPI_DEVICE_INFO_Call_REJECT_CAUSE_USER_REJECT User rejected the call.

QAPI_DEVICE_INFO_Call_REJECT_CAUSE_LOW_BATTERY Call was rejected because of a low battery.

QAPI_DEVICE_INFO_Call_REJECT_CAUSE_BLACKLISTED_CALL_ID Call was rejected because the number was blacklisted.

QAPI_DEVICE_INFO_Call_REJECT_CAUSE_DEAD_BATTERY Call was rejected because of a dead battery.

QAPI_DEVICE_INFO_Call_REJECT_CAUSE_UNWANTED_CALL Call was rejected because the received call was an unwanted robocall.

8.1.4.25 enum qapi_Device_info_Call_State_t

Enumerator:

QAPI_DEVICE_INFO_CALL_STATE_ORIGINATION **QAPI_DEVICE_INFO_CALL_STATE_INCOMING**
QAPI_DEVICE_INFO_CALL_STATE_CONVERSATION
QAPI_DEVICE_INFO_CALL_STATE_CC_IN_PROGRESS Call is originating but waiting for call control to complete.
QAPI_DEVICE_INFO_CALL_STATE_ALERTING **QAPI_DEVICE_INFO_CALL_STATE_HOLD**
QAPI_DEVICE_INFO_CALL_STATE_WAITING **QAPI_DEVICE_INFO_CALL_STATE_DISCONNECTING**
QAPI_DEVICE_INFO_CALL_STATE_END
QAPI_DEVICE_INFO_CALL_STATE_SETUP MT call is in Setup state in 3GPP.

8.1.4.26 enum qapi_Device_info_Call_Direction_t

Voice call redirection.

Enumerator:

QAPI_DEVICE_INFO_CALL_DIRECTION_MO Mobile originating call (originating a call).
QAPI_DEVICE_INFO_CALL_DIRECTION_MT Mobile terminating call (receiving a call).

8.1.4.27 enum qapi_Device_Info_Power_Save_Duration_Inc_Type_t

QAPI Device Info Scan Config Power Inc Type.

Enumerator:

QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_TYPE_ENUM_MIN_ENUM_VAL Forces a 32-bit signed enum. Do not change or use.
QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_STATIC Power save duration remains static.
QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_LINEAR Power save duration is incremented linearly.
QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_EXP Power save duration is incremented exponentially
QAPI_DEVICE_INFO_POWER_SAVE_DURATION_INC_TYPE_ENUM_MAX_ENUM_VAL Forces a 32-bit signed enum. Do not change or use.

8.1.4.28 enum qapi_Device_Info_PDP_Response_code

QAPI Device information data profile error types.

Enumerator:

QAPI_DEVICE_INFO_PDP_NO_ERROR	No PDP error return
QAPI_DEVICE_INFO_PDP_FAIL	Internal failure
QAPI_DEVICE_INFO_PDP_ERR_INVALID_IDENT	Invalid identifier
QAPI_DEVICE_INFO_PDP_ERR_INVALID_ARGS	Invalid arguments
QAPI_DEVICE_INFO_PDP_ERR_OUT_OF_PROFILES	No profile available while create
QAPI_DEVICE_INFO_PDP_ERR_OUT_OF_MEMORY	Out of memory
QAPI_DEVICE_INFO_PDP_ERR_FILE_ACCESS	Error accessing while EFS
QAPI_DEVICE_INFO_PDP_ERR_EOF	Error end of file
QAPI_DEVICE_INFO_PDP_PROFILE_ALREADY_PRESENT	Profile with same IP + APN name is present

8.1.4.29 enum qapi_Device_Info_PDP_Type_e

QAPI Device information PDP type.

Enumerator:

QAPI_DEVICE_INFO_PDP_MIN_VAL	Minimum value
QAPI_DEVICE_INFO_PDP_IPV4_VAL	IPv4 family type
QAPI_DEVICE_INFO_PDP_PPP_VAL PPP	family type
QAPI_DEVICE_INFO_PDP_IPV6_VAL IPv6	family type
QAPI_DEVICE_INFO_PDP_IPV4V6_VAL IPv4v6	family type
QAPI_DEVICE_INFO_PDP_NON_IP_VAL Non_IP	family type
QAPI_DEVICE_INFO_PDP_MAX_VAL	Maximum value

8.1.4.30 enum qapi_Device_Info_Auth_Type_e



QAPI Device information authentication preference type.

Enumerator:

QAPI_DEVICE_INFO_PAP_CHAP_NOT_ALLOWED_E Neither of the authentication protocols (PAP, CHAP) are allowed.

QAPI_DEVICE_INFO_PAP_ONLY_ALLOWED_E Only PAP authentication protocol is allowed.

QAPI_DEVICE_INFO_CHAP_ONLY_ALLOWED_E Only CHAP authentication protocol is allowed.

QAPI_DEVICE_INFO_PAP_CHAP_BOTH_ALLOWED_E Both PAP and CHAP authentication protocols are allowed.

8.1.4.31 enum qapi_Device_Info_PDP_Hdr_Compr_Type_e

QAPI device information PDP header compression type.

Enumerator:

QAPI_DEVICE_INFO_PDP_HDR_COMPR_TYPE_OFF_E PDP header compression is off.

QAPI_DEVICE_INFO_PDP_HDR_COMPR_TYPE_MANUFACTURER_E Manufacturer preferred compression.

QAPI_DEVICE_INFO_PDP_HDR_COMPR_TYPE_RFC_1144_E PDP header compression based on RFC2507.

QAPI_DEVICE_INFO_PDP_HDR_COMPR_TYPE_RFC_2507_E PDP header compression based on RFC1144.

QAPI_DEVICE_INFO_PDP_HDR_COMPR_TYPE_RFC_3095_E PDP header compression based on RFC3095.

8.1.4.32 enum qapi_Device_Info_PDP_Data_Compr_Type_e

QAPI device information PDP data compression type.

Enumerator:

QAPI_DEVICE_INFO_PDP_DATA_COMPR_TYPE_OFF_E PDP data compression is off.

QAPI_DEVICE_INFO_PDP_DATA_COMPR_TYPE_MANUFACTURER_E Manufacturer preferred compression.

QAPI_DEVICE_INFO_PDP_DATA_COMPR_TYPE_V42_E V.42BIS data compression.

QAPI_DEVICE_INFO_PDP_DATA_COMPR_TYPE_V44_E V.44 data compression

8.1.4.33 enum qapi_Device_Info_PLMN_Act_Type_t

Enumerator:

QAPI_DEVICE_INFO_PLMN_ACT_GSM GSM

QAPI_DEVICE_INFO_PLMN_ACT_LTE_M1 LTE Cat M1

QAPI_DEVICE_INFO_PLMN_ACT_LTE_NB1 LTE Cat NB1

8.2 API Functions

8.2.1 qapi_Device_Info_Init_v2

Initializes the device information context.

Prototype

```
qapi_Status_t qapi_Device_Info_Init_v2 ( qapi_Device_Info_Hndl_t *device_info_hdl )
```

Parameters

in	<i>device_info_hdl</i>	Pointer to device information handle.
----	------------------------	---------------------------------------

This function must be called before invoking other qapi_Device_Info APIs.

Returns

- QAPI_OK on success
- QAPI_ERROR on failure.

8.2.2 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_hdl,
qapi_Device_Info_ID_t id, qapi_Device_Info_t * info )
```

Parameters

in	<i>device_info_hdl</i>	The device information handle.
in	<i>id</i>	Information ID.
out	<i>info</i>	Information received for the specified ID.

Returns

- QAPI_OK on success
- QAPI_ERROR on failure.

Dependencies

Before calling this API, [qapi_Device_Info_Init_v2\(\)](#) must have been called.

8.2.3 qapi_Device_Info_Set_Callback_v2

Sets a device information callback and associated information ID.

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

in	<i>device_info_hndl</i>	The device information handle.
in	<i>id</i>	Information ID to be set.
in	<i>callback</i>	Callback to be registered.

Returns

- QAPI_OK on success
- QAPI_ERROR on failure.

Dependencies

Before calling this API, [qapi_Device_Info_Init_v2\(\)](#) must have been called.

8.2.4 qapi_Device_Info_Release_v2

Releases the device information context.

in	<i>device_info_hdl</i>	The device information handle.
----	------------------------	--------------------------------

Prototype

```
qapi_Status_t qapi_Device_Info_Release_v2( qapi_Device_Info_Hndl_t device_info_hdl)
```

Returns

- QAPI_OK on success
- QAPI_ERROR on failure.

Dependencies

Before calling this API, [qapi_Device_Info_Init_v2\(\)](#) must have been called.

8.2.5 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

in	<i>device_info_hndl</i>	The device information handle.
----	-------------------------	--------------------------------

Returns

- QAPI_OK on success
- QAPI_ERROR on failure.

8.2.6 qapi_Device_Info_Request

Requests device information for specified ID.

Prototype

```
qapi_Status_t qapi_Device_Info_Request ( qapi_Device_Info_Hndl_t device_info_hdl,  
qapi_Device_Info_Req_ID_t id, qapi_Device_Info_Request_t *req, qapi_Device_Info_Response_t *  
rsp )
```

Parameters

in	<i>device_info_hdl</i>	The device information handle.
in	<i>id</i>	Device information request type.
in	<i>req</i>	Device information request structure that sets parameters based on request ID.
in,out	<i>rsp</i>	Device information response structure that fetches parameters based on request ID.

Returns

- QAPI_OK on success
- QAPI_ERROR on failure.



Note: The device should be in low power mode with no available subscription for the following IDs.

Otherwise, QAPI returns "QAPI_ERR_INVALID_STATE" error.

- QAPI_DEVICE_INFO_REQ_FSK_START_E
- QAPI_DEVICE_INFO_REQ_FSK_STOP_E
- QAPI_DEVICE_INFO_REQ_GET_FSK_DEBUG_E
- QAPI_DEVICE_INFO_REQ_SET_FSK_DEBUG_E
- QAPI_DEVICE_INFO_REQ_GET_FSK_PCL_E
- QAPI_DEVICE_INFO_REQ_SET_FSK_PCL_E

Dependencies

Before calling this function, [qapi_Device_Info_Init_v2\(\)](#) must be called.

FIBOCOM
Confidential

8.2.7 qapi_Device_Info_Clear_Callback_v2

Clears a particular indication associated with a handle.

Prototype

```
qapi_Status_t qapi_Device_Info_Clear_Callback_v2 ( qapi_Device_Info_ - Hndl_t device_info_hndl,
qapi_Device_Info_ID_t id )
```

Parameters

in	<i>device_info_hndl</i>	The device information handle.
in	<i>id</i>	Information ID to be cleared.

Returns

- QAPI_OK on success
- QAPI_ERROR on failure.

Dependencies

Before calling this function, [qapi_Device_Info_Init_v2\(\)](#) must be called. This function clears the indication set by using [qapi_Device_Info_Set_Callback_v2\(\)](#).

9 GPIO Interrupt Controller APIs

The general purpose input/output (GPIO) interrupt controller provides an interface for registering for interrupts for a GPIO. These are generally used for customer-specific use cases in which an entity external to the chip needs to communicate with the chip. 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.

- * The code snippet below demonstrates the use of this interface. The
- * example below includes the `qapi_gpioint.h` header file. This example
- * registers a callback with the GPIO Interrupt driver and manually
- * triggers the interrupt. Although this is a manual trigger use-case,
- * in practice, the GPIO is usually triggered externally to the chip.
- * After triggering the interrupt, it will loop 1000 times and deregister
- * the callback from the driver.
- * This code snippet registers for GPIO 10 specifically and registers
- * the callback that will be defined as type `qapi_GPIOINT_CB_t`.
- * The code registers medium priority. It will be a level high trigger
- * given the input parameter `GPIOINT_TRIGGER_HIGH_LEVEL`, meaning that
- * when the external signal is high, it will jump to the handler if
- * enabled.

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](#)

9.1 GPIO Data Types

9.1.1 Data Typedef

9.1.1.1 typedef uint32_t qapi_GPIOINT_Callback_Data_t

GPIO interrupt callback data type.

This is the data type of the argument passed into the callback that is registered with the GPIO interrupt module. The value to pass will be given by the client at registration time.

9.1.1.2 typedef void(* qapi_GPIOINT_CB_t)(qapi_GPIOINT_Callback_Data_t)

GPIO interrupt callback function definition.

GPIO interrupt clients will pass a function pointer of this format into the registration API.

9.1.1.3 typedef void* qapi_Instance_Handle_t

GPIO interrupt handle definition.

9.1.2 Data Enumeration

9.1.2.1 enum qapi_GPIINT_Trigger_e

GPIO interrupt trigger type enumeration for supported triggers.

Enumerator:

QAPI_GPIINT_TRIGGER_LEVEL_HIGH_E	<i>Level triggered active high.</i>
QAPI_GPIINT_TRIGGER_LEVEL_LOW_E	<i>Level triggered active low.</i>
QAPI_GPIINT_TRIGGER_EDGE_RISING_E	<i>Rising edge triggered.</i>
QAPI_GPIINT_TRIGGER_EDGE_FALLING_E	<i>Falling edge triggered.</i>
QAPI_GPIINT_TRIGGER_EDGE_DUAL_E	<i>Dual edge triggered.</i>

9.1.2.2 enum qapi_GPIINT_Priority_e

GPIO interrupt priority selection. The priority can determine how the interrupt is configured internally.

Enumerator:

QAPI_GPIINT_PRIO_HIGHEST_E	<i>Highest priority</i>
QAPI_GPIINT_PRIO_HIGH_E	<i>Medium-high priority</i>
QAPI_GPIINT_PRIO_MEDIUM_E	<i>Medium priority</i>
QAPI_GPIINT_PRIO_LOW_E	<i>Medium-low priority</i>
QAPI_GPIINT_PRIO_LOWEST_E	<i>Highest priority</i>

9.2 API Functions

9.2.1 qapi_GPIOINT_Register_Interrupt

Registers a callback for a GPIO interrupt.

Registers a callback function with the GPIO interrupt controller and enables 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_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

in	<i>pH</i>	Input handle to the client context.
in	<i>nGpio</i>	GPIO number to configure for an interrupt.
in	<i>pfnCallback</i>	Callback function pointer.
in	<i>nData</i>	Callback data.
in	<i>eTrigger</i>	Trigger type for the interrupt.
in	<i>ePriority</i>	Priority enumeration to determine the configuration of the GPIO interrupt.
in	<i>bNmi</i>	Boolean value to select whether or not the GPIO interrupt is nonmaskable to the CPU.

Returns

- QAPI_ERR_INVALID_PARAM – There is an issue with one of the input parameters.
- QAPI_ERROR – Error in internal registration.
- QAPI_OK – Successfully registered.



Note: QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

FIBOCOM
Confidential

9.2.2 qapi_GPIOINT_Deregister_Interrupt

Deregisters a callback function from the GPIO interrupt controller and disables the interrupt. This function deconfigures the interrupt accordingly, as well as disabling it in the underlying layers.

Prototype

```
qapi_Status_t qapi_GPIOINT_Deregister_Interrupt ( qapi_Instance_Handle_t *pH, uint32_t nGpio )
```

Parameters

<i>in</i> <i>pH</i>	Input handle to the client context.
<i>in</i> <i>nGpio</i>	GPIO number to deconfigure.

Returns

- QAPI_ERROR – Error in internal deregistration.
- QAPI_OK – Successfully deregistered.



Note:

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

9.2.3 qapi_GPIOINT_Set_Trigger

Dynamically sets the trigger type of a registered GPIO interrupt.

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 eTrigger )
```

Parameters

in	<i>pH</i>	Input handle to the client context.
in	<i>nGpio</i>	GPIO number in which to set the trigger.
in	<i>eTrigger</i>	Trigger type to configure.

Returns

- QAPI_ERR_INVALID_PARAM – eTrigger parameter is invalid.
- QAPI_ERROR – Internal error in setting trigger.
- QAPI_OK – Successfully set the trigger.



Note:

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

9.2.4 qapi_GPIOINT_Enable_Interrupt

Enables 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

<i>in</i> <i>pH</i>	Input handle to the client context.
<i>in</i> <i>nGpio</i>	GPIO number to enable.

Returns

- QAPI_ERROR – Internal error in enabling interrupt.
- QAPI_OK – Successfully enabled interrupt.



Note:

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

9.2.5 qapi_GPIOINT_Disable_Interrupt

Disables 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

<i>in</i> pH	Input handle to the client context.
<i>in</i> nGpio	GPIO number to disable.

Returns

- QAPI_ERROR – Internal error in disabling interrupt.
- QAPI_OK – Successfully disabled interrupt.



Note:

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

9.2.6 qapi_GPIOINT_Trigger_Interrupt

Manually triggers 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

<i>in</i> pH	Input handle to the client context.
<i>in</i> nGpio	GPIO number to trigger.

Returns

- QAPI_ERROR – Internal error in triggering interrupt.
- QAPI_OK – Successfully triggered interrupt.



Note:

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

9.2.7 qapi_GPIOINT_Is_Interrupt_Pending

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 * pblsPending )
```

Parameters

in	<i>pH</i>	Input handle to the client context.
in	<i>nGpio</i>	GPIO number to trigger.
out	<i>pblsPending</i>	Boolean value for whether or not the interrupt is pending in hardware.

Returns

- QAPI_ERR_INVALID_PARAM – pblsPending pointer is NULL.
- QAPI_ERROR – Internal error in checking pending.
- QAPI_OK – Successfully checked pending status.



Note:

QAPI_ERROR may be returned if there is an invalid handle or an incorrect or invalid GPIO is being used.

10 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.

The code snippet below shows an example usage of the programming interface. The module requires clients to use physical pin numbers on the SoC. Consult the hardware schematic or use a device configuration database to determine the proper pin number.

- * The code snippet below demonstrates usage of the PMM interface. The
- * example below configures SoC pin-13 to be a discrete GPIO configured
- * as an input with a pull-down. Note that drive strength is defaulted
- * to be QAPI_GPIO_2MA_E, even though it is not applicable for pins
- * configured as discrete inputs.

This chapter provides the following QAPIs:

- [qapi_TLMM_Get_Gpio_ID](#)
- [qapi_TLMM_Release_Gpio_ID](#)
- [qapi_TLMM_Config_Gpio](#)

10.1 PMM Date Types

10.1.1 Data Structure

10.1.1.1 struct qapi_TLMM_Config_t

GPIO configuration.

This structure specifies the configuration for a GPIO on the SoC. The GPIO can be configured as input or output, which can be driven high or low by the software. The interface also allows the SoC pins to be configured for alternate functionality.

Data fields

Type	Parameter	Description
uint32_t	pin	Physical pin number.
uint32_t	func	Pin function select.
qapi_GPIO_ Direction_t	dir	Direction (input or output).
qapi_GPIO_ Pull_t	pull	Pull value.
qapi_GPIO_ Drive_t	drive	Drive strength.

10.1.2 Data Typedef

10.1.2.1 `typedef uint16_t qapi_GPIO_ID_t`

SoC pin access ID.

The module provides the unique ID to the client. Clients must pass this ID as a token with subsequent calls. Note that clients should cache the ID.

10.1.3 Data Enumeration

10.1.3.1 enum qapi_GPIO_Direction_t

Pin direction enumeration.

This enumeration specifies the direction when configuring a GPIO pin.

Enumerator:

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.

10.1.3.2 enum qapi_GPIO_Pull_t

GPIO pin pull type.

This enumeration specifies the type of pull (if any) to use when specifying the configuration for a GPIO pin.

Enumerator:

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.

10.1.3.3 enum qapi_GPIO_Drive_t

GPIO pin drive strength.

This enumeration specifies the drive strength to use when specifying the configuration of a GPIO pin.

Enumerator:

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 an 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.

10.1.3.4 enum qapi_GPIO_Value_t

GPIO output state specification.

This enumeration specifies the value to write to a GPIO pin configured as output. This functionality is also known as *bit banging*.

Enumerator:

QAPI_GPIO_LOW_VALUE_E Drive the output LOW.

QAPI_GPIO_HIGH_VALUE_E Drive the output HIGH.

10.2 API Functions

10.2.1 qapi_TLMM_Get_Gpio_ID

Gets a unique access 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

in	<i>qapi_TLMM_Config</i>	Pointer to the pin configuration data.
in	<i>qapi_GPIO_ID</i>	Pointer to a location in which to store the access ID.

Returns

QAPI_OK – Pin GPIO ID was successfully created.

QAPI_ERR – Pin GPIO is currently in use or not programmable.

10.2.2qapi_TLMM_Release_Gpio_ID

Releases an SoC pin.

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. Using this function is not required 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

in	<i>qapi_TLMM_Config</i>	Pointer to pin configuration data.
in	<i>qapi_GPIO_ID</i>	Pin access ID retrieved from the qapi_TLMM_Get_Gpio_ID() call

Returns

QAPI_OK – Pin was released successfully

QAPI_ERR – Pin could not be released

10.2.3 qapi_TLMM_Config_Gpio

Changes the SoC pinconfiguration

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

in	<i>qapi_GPIO_ID</i>	Pin access ID retrieved from the qapi_TLMM_Get_Gpio_ID() call
in	<i>qapi_TLMM_Config</i>	Pin configuration to use.

Returns

QAPI_OK – Pin was configured successfully
QAPI_ERR – Pin could not be configured

11 Pulse Width Modulation (PWM)

Many peripherals are controlled through PWM signals. For example, an LED's brightness, a motor's speed, a buzzer and many more.

This module exposes an interface allowing its clients to configure the PWM block in the SoC. The configuration consists of setting the frequency and the duty cycle of the PWM output signal.

In the SoC, there can be multiple PWM frames, with each frame having multiple programmable instances. Please refer to the chipset's documentation for details pertaining to available PWM frames & instances.

- * The code snippet below demonstrates usage of the PWM interface. The
- * example below configures SoC PWM instance-0 to be generate a signal
- * with 25% duty cycle and operating at 960 KHz.
- * For this example, the PWM core clock frequency is assumed to be
- * configured to 19.2 MHz.
- * Clients can invoke `qapi_PWM_Get_Clock_Frequency` to obtain the frequency.

This chapter provides the following QAPIs:

- [qapi_PWM_Get_ID](#)
- [qapi_PWM_Release_ID](#)
- [qapi_PWM_Enable](#)
- [qapi_PWM_Set_Frequency](#)
- [qapi_PWM_Set_Duty_Cycle](#)
- [qapi_PWM_Get_Clock_Frequency](#)
- [qapi_TLMM_Drive_Gpio](#)
- [qapi_TLMM_Read_Gpio](#)

11.1 PWM Data Types

11.1.1 Data Typedef

11.1.1.1 typedef uint32_t qapi_PWM_ID_t

SoC PWM access ID.

Unique ID provided by the module to the client. Clients must pass this ID as a token with subsequent calls. Note that clients should cache the ID.

11.1.2 Data Enumeration

1.1.1.1 enum qapi_PWM_Frame_t

PWM frame enumeration.

The enumeration is used to get the ID to specific PWM frame's instance within the SoC.

1.1.1.2 enum qapi_PWM_Instance_t

PWM instance enumeration.

The enumeration is used to get the ID to a specific PWM frame's instance within the SoC.

11.2 API Functions

11.2.1 qapi_PWM_Get_ID

Gets a unique access ID.

This function provides a unique access ID for a specified PWM instance. This is required in order to access PWM configuration APIs.

Prototype

```
qapi_Status_t qapi_PWM_Get_ID ( qapi_PWM_Frame_t eFrame, qapi_PWM_Instance_t eInstance,
qapi_PWM_ID_t * pnID )
```

Parameters

<i>in eFrame</i>	PWM Frame.
<i>in eInstance</i>	PWM instance to get the ID for.
<i>in pnID</i>	Pointer to a location in which to store the access ID.

Returns

- QAPI_OK – PWM ID was successfully created.
- QAPI_ERR – PWM ID is currently in use or invalid instance was requested.

11.2.2qapi_PWM_Release_ID

Releases a SoC PWM instance.

This function allows a client to relinquish the lock on a PWM instance. It facilitates sharing of a PWM instance between two drivers in different system modes where each driver may need to reconfigure the PWM instance. Using this function is not required unless such a condition dictates.

Prototype

```
qapi_Status_t qapi_PWM_Release_ID ( qapi_PWM_ID_t nID )
```

Parameters

in	<i>nID</i>	PWM instance access ID retrieved from the qapi_PWM_Get_ID() call.
----	------------	---

Returns

- QAPI_OK – ID was released successfully
- QAPI_ERR – ID could not be released.

11.2.3qapi_PWM_Enable

Enables or disables a SoC PWM instance.

This function enable the clock(s) required for enabling the requested PWM instance and also enables the same.

It facilitates sharing of a PWM instance between two drivers in different system modes where each driver may need to reconfigure the PWM instance. Using this function is not required unless such a condition dictates.

Prototype

```
qapi_Status_t qapi_PWM_Enable ( qapi_PWM_ID_t nID, qbool_t bEnable )
```

Parameters

<i>in</i> nID	PWM instance access ID retrieved from the qapi_PWM_Get_ID() call.
<i>in</i> bEnable	Boolean flag for enable (TRUE) or disable (FALSE) request

Returns

- QAPI_OK – PWM instance was enabled successfully
- QAPI_ERR – PWM instance was not enabled

11.2.4qapi_PWM_Set_Frequency

Sets the frequency of the output PWM signal.

This function configures the frequency (period) of the requested PWM ID's signal.



Note:

The frequency can only be set once owing to hardware limitations. This API needs to be called before setting the duty cycle.

The PWM output frequency range (outlined below) is determined by the PWM core clock frequency. Clients can invoke `qapi_PWM_Get_Clock_Frequency` to obtain the frequency and check if it meets their requirements.

Min PWM output frequency is $(\text{PWM clock frequency}) / 2^{16}$. Max PWM output frequency is $(\text{PWM clock frequency}) / 2$.

For example, if the PWM clock frequency is 19.2 MHz, then: min 292 Hz $(19.2 \text{ MHz} / 2^{16})$, max 9.6 MHz $(4.8 / 2)$.

Prototype

```
qapi_Status_t qapi_PWM_Set_Frequency ( qapi_PWM_ID_t nID, uint32_t nFreqHz )
```

Parameters

<code>nID</code>	ID retrieved from the <code>qapi_PWM_Get_ID()</code> call.
<code>nFreqHz</code>	Frequency of the PWM signal in Hz to set to.

Returns

- QAPI_OK – Frequency was configured successfully
- QAPI_ERR – Frequency could not be configured.

Some of the reasons this could happen are:

- 1) Client does not have ID for PWM instance
- 2) Requested frequency is not within range
- 3) Client tried to change the frequency

FIBOCOM
Confidential

11.2.5qapi_PWM_Set_Duty_Cycle

This function configures the duty cycle (active period) of the requested PWM ID's signal.



Note:

The duty cycle can only be set to quantized levels. The specific levels depend upon the PWM output frequency and the PWM core clock frequency. Clients are responsible to check this and request the duty cycle accordingly. If the step size is not an integer, then use the rounded up integer duty cycle.

For example, if step size is 6.25%, then 12.5%, 18.75% can be supported. Client needs to pass 19% as the parameter for 18.75%. Client needs to pass 13% as the parameter for 12.5%. Client needs to pass 7% as the parameter for 6.25%.

If client passes a duty cycle that cannot be achieved, then the duty cycle will be set to the max achievable duty cycle lesser than the requested one. As per above example, if 21% is passed, then 18.75% would be set.

Duty cycle step size in %: $100 \times (\text{PWM output frequency}) / (\text{PWM clock frequency})$

For example, if PWM clock frequency is 19.2 MHz and output is 1.92 MHz, then the duty cycle step size will be: $100 \times 1.92 / 19.2 = 10\%$ Hence, only duty cycles of 10%, 20%, 30% and so on can be obtained. In this example, for 1% step size, the user has to set the output frequency to 48 KHz or lower.

The PWM clock frequency can be obtained from `qapi_PWM_Get_Clock_Frequency`.

Prototype

```
qapi_Status_t qapi_PWM_Set_Duty_Cycle ( qapi_PWM_ID_t nID, uint32_t nDuty )
```

Parameters

<code>in nID</code>	ID retrieved from the <code>qapi_PWM_Get_ID()</code> call.
<code>in nDuty</code>	Duty cycle of the PWM signal in % (1-99)

Returns

- QAPI_OK – Duty cycle was configured successfully
- QAPI_ERR – Duty cycle could not be configured

Some of the reasons this could happen are:

- a) Client does not have ID for PWM instance
- b) Frequency has not been set
- c) Requested duty cycle is lesser than the least achievable one

FIBOCOM
Confidential

11.2.6 qapi_PWM_Get_Clock_Frequency

Get the PWM clock frequency of a PWM instance.

This function queries the PWM core clock frequency so that clients can ascertain the frequency range and duty cycle step size.

Prototype

```
qapi_Status_t qapi_PWM_Get_Clock_Frequency ( qapi_PWM_ID_t nID, uint32_t * pnFreqHz )
```

Parameters

in	<i>nID</i>	Pin access ID retrieved from the qapi_PWM_Get_ID() call.
out	<i>pnFreqHz</i>	Pointer to fill in with the frequency. A value of zero indicates that the frequency could not be determined.

Returns

- QAPI_OK – Operation completed successfully
- QAPI_ERR – Operation failed

11.2.7 qapi_TLMM_Drive_Gpio

Sets the state of an SoC pin configured as an output GPIO.

This function drives the output of an 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

in	<i>qapi_GPIO_ID</i>	Pin access ID retrieved from the qapi_TLMM_Get_Gpio_ID() call
in	<i>pin</i>	SoC pin number to configure.
in	<i>value</i>	Output value.

Returns

- QAPI_OK – Operation completed successfully
- QAPI_ERR – Operation failed

11.2.8qapi_TLMM_Read_Gpio

Reads the state of an 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

in	<i>qapi_GPIO_ID</i>	Pin access ID retrieved from the qapi_TLMM_Get_Gpio_ID() cal
in	<i>pin</i>	SoC pin number to configure.
out	<i>qapi_GPIO_Value</i>	GPIO pin value.

Returns

- QAPI_OK – Operation completed successfully
- QAPI_ERR – Operation failed

12 I2C Master 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](#)

12.1 I2C Data Types

12.1.1 Data Define

12.1.1.1 #define QAPI_I2C_FLAG_START 0x00000001

Specifies that the transfer begins with a START bit - S.

12.1.1.2 #define QAPI_I2C_FLAG_STOP 0x00000002

Specifies that the transfer ends with a STOP bit - P.

12.1.1.3 #define QAPI_I2C_FLAG_WRITE 0x00000004

Must be set to indicate a WRITE transfer.

12.1.1.4 #define QAPI_I2C_FLAG_READ 0x00000008

Must be set to indicate a READ transfer.

12.1.1.5 #define QAPI_I2C_TRANSFER_MASK (QAPI_I2C_FLAG_WRITE | QAPI_I2C_FLAG_READ)

Transfer types.

12.1.1.6 #define QAPI_VALID_FLAGS(x) (((x & QAPI_I2C_TRANSFER_MASK) == QAPI_I2C_FLAG_READ) || ((x & QAPI_I2C_TRANSFER_MASK) == QAPI_I2C_FLAG_WRITE))

Verifies the validity of flags.

12.1.2 Data Structure

12.1.2.1 struct qapi_I2CM_Config_t

I2C client configuration parameters that the client uses to communicate to an I2C slave.

Data fields

Type	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 us that a slave might perform.
uint32_t	core_- Configuration1	Core specific configuration. Recommended is 0.
uint32_t	core_- Configuration2	Core specific configuration. Recommended is 0.

12.1.2.2 struct qapi_I2CM_Descriptor_t

I2C transfer descriptor.

Data fields

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.

FIBOCOM
Confidential


12.1.3Data Typedef

12.1.3.1 typedef void(* qapi_I2CM_Transfer_CB_t)(const uint32_t status, void*CB_Parameter)

Transfer callback.

Declares 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.

 **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.

Parameters

out	<i>Status</i>	Completion status of the transfer. A call to qapi_I2CM_Get_QStatus_Code() will convert this status to QAPI status codes.
out	<i>CB_Parameter</i>	CP_Parameter context that was passed in the call to qapi_I2CM_Transfer() .

12.1.4 Data Enumeration

12.1.4.1 enum qapi_I2CM_Instance_t

Instance of the I2C core that the client wants to use. This instance is passed in [qapi_I2CM_Open\(\)](#).

Enumerator:

QAPI_I2CM_INSTANCE_1_E	<i>I2C core01</i>
QAPI_I2CM_INSTANCE_2_E	<i>I2C core02</i>
QAPI_I2CM_INSTANCE_3_E	<i>I2C core03</i>
QAPI_I2CM_INSTANCE_4_E	<i>I2C core04</i>
QAPI_I2CM_INSTANCE_5_E	<i>I2C core05</i>
QAPI_I2CM_INSTANCE_6_E	<i>I2C core06</i>
QAPI_I2CM_INSTANCE_7_E	<i>I2C core07</i>
QAPI_I2CM_INSTANCE_8_E	<i>I2C core08</i>
QAPI_I2CM_INSTANCE_9_E	<i>I2C core09</i>
QAPI_I2CM_INSTANCE_10_E	<i>I2C core10</i>
QAPI_I2CM_INSTANCE_11_E	<i>I2C core11</i>
QAPI_I2CM_INSTANCE_12_E	<i>I2C core12</i>
QAPI_I2CM_INSTANCE_13_E	<i>I2C core13</i>
QAPI_I2CM_INSTANCE_14_E	<i>I2C core14</i>
QAPI_I2CM_INSTANCE_15_E	<i>I2C core15</i>
QAPI_I2CM_INSTANCE_16_E	<i>I2C core16</i>
QAPI_I2CM_INSTANCE_17_E	<i>I2C core17</i>
QAPI_I2CM_INSTANCE_18_E	<i>I2C core18</i>
QAPI_I2CM_INSTANCE_19_E	<i>I2C core19</i>
QAPI_I2CM_INSTANCE_20_E	<i>I2C core20</i>
QAPI_I2CM_INSTANCE_21_E	<i>I2C core21</i>



QAPI_I2CM_INSTANCE_22_E *I2C core22*

QAPI_I2CM_INSTANCE_23_E *I2C core23*

QAPI_I2CM_INSTANCE_24_E *I2C core24*

FIBOCOM
Confidential

12.2 API Functions

12.2.1 qapi_I2CM_Open

Called by the client code to initialize the respective I2C instance. On success, `i2c_Handle` points to the handle for the I2C instance. The API allocates resources for use by the client handle and the I2C instance. These resources are release in the [qapi_I2CM_Close\(\)](#) call. The API also enables power to the I2C hardware 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 **i2c_Handle )
```

Parameters

in	<i>instance</i>	I2C instance that the client intends to initialize; see qapi_I2CM_Instance_t for details.
out	<i>i2c_Handle</i>	Pointer location to be filled by the driver with a handle to the instance.

Returns

- QAPI_OK – Function was successful.
- QAPI_I2CM_ERR_UNSUPPORTED_CORE_INSTANCE – Unsupported core instance.
- QAPI_I2CM_ERR_HANDLE_ALLOCATION – Handle allocation error.
- QAPI_I2CM_ERROR_OPEN_FAILURE – Failure in Open.

12.2.2qapi_I2CM_Close

De-initializes the I2C instance and releases any resources allocated by the [qapi_I2CM_Open\(\)](#) API.

Prototype

```
qapi_Status_t qapi_I2CM_Close ( void * i2c_Handle )
```

Parameters

in	<i>i2c_Handle</i>	Handle to the I2C instance.
----	-------------------	-----------------------------

Returns

- QAPI_OK – Function was successful.
- QAPI_I2CM_ERROR_CLOSE_FAILURE – Failure in Close.
- QAPI_I2CM_ERROR_PLATFORM_DETACH_FAILURE – Failure to detach with DAL layer.

Dependencies

- Before calling this API [qapi_I2CM_Open\(\)](#) and [qapi_I2CM_Power_On](#) API must be called.

12.2.3qapi_I2CM_Transfer

Performs 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.



Note :

After a client calls this API, it must wait for the completion callback to occur before it can call the API again. If the client wishes to queue multiple transfers, it must use an array of descriptors of type [qapi_I2CM_Descriptor_t](#) instead of calling the API multiple times.

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

in	<i>i2c_Handle</i>	Handle to the I2C instance.
in	<i>config</i>	Slave configuration. See qapi_I2CM_Config_t for details.
in	<i>desc</i>	I2C transfer descriptor. See qapi_I2CM_Descriptor_t for details. This can be an array of descriptors.
in	<i>num_Descriptors</i>	Number of descriptors in the descriptor array.
in	<i>CB_Function</i>	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.
in	<i>CB_Parameter</i>	Context that the client passes here is returned as is in the callback function.
in	<i>delay_us</i>	A delay in microseconds that specifies the time to wait before

		starting the transfer.
--	--	------------------------

Returns

- QAPI_OK – Function was successful.
- QAPI_I2CM_ERR_INVALID_PARAMETER – Invalid parameter.

Dependencies

Before calling this API [qapi_I2CM_Open\(\)](#) and [qapi_I2CM_Power_On\(\)](#) API must be called.

12.2.4 qapi_I2CM_Power_On

Enables the I2C Hardware resources for an I2C transaction.

This function enables all resources required for a successful I2C transaction. This includes clocks, power resources and pin multiplex functions. This function should be called before a transfer or a batch of I2C transfers.

Parameters

in	<i>i2c_Handle</i>	Driver handle returned by qapi_I2CM_Open() .
----	-------------------	--

Returns

- QAPI_OK – I2C master enabled successfully.
- QAPI_I2CM_ERR_INVALID_PARAMETER – Invalid handle parameter.
- QAPI_I2CM_ERROR_POWER_ON_FAILURE – Failure in Power On.

Dependencies

- Before calling this API [qapi_I2CM_Open\(\)](#) API must be called.

12.2.5 qapi_I2CM_Power_Off

Disables the I2C Hardware resources for an I2C transaction.

This function turns off all resources used by the I2C master. This includes clocks, power resources and GPIOs. This function should be called to put the I2C master in its lowest possible power state.

Prototype

```
qapi_Status_t qapi_I2CM_Power_Off ( void * i2c_Handle )
```

Parameters

in	<i>i2c_Handle</i>	Driver handle returned by qapi_I2CM_Open() .
----	-------------------	--

Returns

- QAPI_OK – I2C master disabled successfully.
- QAPI_I2CM_ERR_INVALID_PARAMETER – Invalid handle parameter.
- QAPI_I2CM_ERROR_POWER_OFF_FAILURE – Failure in Power Off.

Dependencies

- Before calling this API [qapi_I2CM_Open\(\)](#) and [qapi_I2CM_Power_On\(\)](#) API must be called.

13 SPI Master 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.

Typical usage:

- [qapi_SPIM_Open\(\)](#) – Get a handle to an SPI instance.
- [qapi_SPIM_Power_On\(\)](#) – Turn on all resources required for a successful SPI transaction.
- [qapi_SPIM_Full_Duplex\(\)](#) – Generic transfer API to perform a transfer on the SPI bus.
- [qapi_SPIM_Power_Off\(\)](#) – Turn off all resources set by [qapi_SPIM_Power_On\(\)](#).
- [qapi_SPIM_Close\(\)](#) – Destroy all objects created by the SPI handle. A note about SPI power:

Calling [qapi_SPIM_Open\(\)](#) and leaving it open does not drain any power. If the client is expecting to do several back-to-back SPI transfers, the recommended approach is to call [Power_On](#), perform all transfers, then call [Power_Off](#). Calling [Power_On](#)/[Power_Off](#) for every transfer will affect throughput and increase the bus idle period.

SPI transfers:

SPI transfers use BAM (DMA mode), so we expect buffers passed by the client to be uncached RAM addresses. There is no address or length alignment requirement.

SPI modes:

The SPI master supports all four SPI modes, and this can be changed per transfer. See [qapi_SPIM_Config_t](#) for configuration specification details. The driver supports parallel transfers on different SPI instances.

A note about SPI modes:

Always check the meaning of SPI modes in your SPI slave specifications. Some manufacturers use different mode meanings.

- SPI Mode 0: CPOL = 0, and CPHA = 0
- SPI Mode 1: CPOL = 0, and CPHA = 1
- SPI Mode 2: CPOL = 1, and CPHA = 0
- SPI Mode 3: CPOL = 1, and CPHA = 1

13.1 SPI Data Types

13.1.1 Data Structure

13.1.1.1 struct qapi_SPIM_Config_t

SPI master configuration.

The SPI master configuration is the collection of settings specified for each SPI transfer call to select the various possible SPI transfer parameters.

Data fields

Type	Parameter	Description
qapi_SPIM_- Shift_Mode_t	SPIM_Mode	SPIM mode type to be used for the SPI core.
qapi_SPIM_C- S_Polarity_t	SPIM_CS_- Polarity	CS polarity type to be used for the SPI core.
qapi_SPIM_- Byte_Order_t	SPIM_- endianness	
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.
uint8_t	CS_Clk_Delay-	Number of clock cycles to wait after asserting CS before starting

	_Cycles	transfer.
uint8_t	Inter_Word_- Delay_Cycles	Number of clock cycles to wait between SPI words.
qapi_SPIM_C- S_Mode_t	SPIM_CS_- Mode	CS mode to be used for the SPI core.
qbool_t	loopback_- Mode	Normally 0. If set, the SPI controller will enable Loopback mode; used primarily for testing.

13.1.1.2 struct qapi_SPIM_Descriptor_t

SPI transfer type.

This type specifies the address and length of the buffer for an SPI transaction.

Data fields

Type	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.

13.1.2 Data Typedef

13.1.2.1 `typedef void(* qapi_SPIM_Callback_Fn_t)(uint32_t status, void *callback_Ctxt)`

SPI callback function type.

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 signaled.

13.1.3 Data Enumeration

13.1.3.1 enum qapi_SPIM_Instance_t

SPI instance enumeration.

This enumeration lists the possible SPI instance indicating which HW SPI master is to be used for the current SPI transaction.

Enumerator:

QAPI_SPIM_INSTANCE_1_E	<i>SPIM instance 1</i>
QAPI_SPIM_INSTANCE_2_E	<i>SPIM instance 2</i>
QAPI_SPIM_INSTANCE_3_E	<i>SPIM instance 3</i>
QAPI_SPIM_INSTANCE_4_E	<i>SPIM instance 4</i>
QAPI_SPIM_INSTANCE_5_E	<i>SPIM instance 5</i>
QAPI_SPIM_INSTANCE_6_E	<i>SPIM instance 6</i>
QAPI_SPIM_INSTANCE_7_E	<i>SPIM instance 7</i>
QAPI_SPIM_INSTANCE_8_E	<i>SPIM instance 8</i>
QAPI_SPIM_INSTANCE_9_E	<i>SPIM instance 9</i>
QAPI_SPIM_INSTANCE_10_E	<i>SPIM instance 10</i>
QAPI_SPIM_INSTANCE_11_E	<i>SPIM instance 11</i>
QAPI_SPIM_INSTANCE_12_E	<i>SPIM instance 12</i>
QAPI_SPIM_INSTANCE_13_E	<i>SPIM instance 13</i>
QAPI_SPIM_INSTANCE_14_E	<i>SPIM instance 14</i>
QAPI_SPIM_INSTANCE_15_E	<i>SPIM instance 15</i>
QAPI_SPIM_INSTANCE_16_E	<i>SPIM instance 16</i>
QAPI_SPIM_INSTANCE_17_E	<i>SPIM instance 17</i>
QAPI_SPIM_INSTANCE_18_E	<i>SPIM instance 18</i>
QAPI_SPIM_INSTANCE_19_E	<i>SPIM instance 19</i>
QAPI_SPIM_INSTANCE_20_E	<i>SPIM instance 20</i>

QAPI_SPIM_INSTANCE_21_E	<i>SPIM instance 21</i>
QAPI_SPIM_INSTANCE_22_E	<i>SPIM instance 22</i>
QAPI_SPIM_INSTANCE_23_E	<i>SPIM instance 23</i>
QAPI_SPIM_INSTANCE_24_E	<i>SPIM instance 24</i>

13.1.3.2 enum qapi_SPIM_Shift_Mode_t

SPI phase type.

This type defines the clock phase that the client can set in the SPI configuration.

Enumerator:

QAPI_SPIM_MODE_0_E	<i>CPOL = 0, CPHA = 0</i>
QAPI_SPIM_MODE_1_E	<i>CPOL = 0, CPHA = 1</i>
QAPI_SPIM_MODE_2_E	<i>CPOL = 1, CPHA = 0</i>
QAPI_SPIM_MODE_3_E	<i>CPOL = 1, CPHA = 1</i>

13.1.3.3 enum qapi_SPIM_CS_Polarity_t

SPI chip select polarity type.

Enumerator:

QAPI_SPIM_CS_ACTIVE_LOW_E	<i>During Idle state, the CS line is held low.</i>
QAPI_SPIM_CS_ACTIVE_HIGH_E	<i>During Idle state, the CS line is held high.</i>

13.1.3.4 enum qapi_SPIM_Byte_Order_t

Order in which bytes from Tx/Rx buffer words are put on the bus.

Enumerator:

SPI_NATIVE	<i>Native</i>
SPI_LITTLE_ENDIAN	<i>Little Endian</i>
SPI_BIG_ENDIAN	<i>Big Endian (network)</i>

13.1.3.5 enum qapi_SPIM_CS_Mode_t

SPI chip select assertion type.

This type defines how the chip select line is configured between N word cycles.

Enumerator:

QAPI_SPIM_CS_DEASSERT_E	<i>CS is deasserted after transferring data for N clock cycles.</i>
QAPI_SPIM_CS_KEEP_ASSERTED_E	<i>CS is asserted as long as the core is in the Run state.</i>

13.2 API Functions

13.2.1 `qapi_Status_t qapi_SPIM_Open (qapi_SPIM_Instance_t instance, void ** spi_Handle)`

Initializes the SPI Master.

This function initializes 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

in	<i>instance</i>	SPI instance specified by qapi_SPIM_Instance_t .
out	<i>spi_Handle</i>	Pointer to a location in which to store the driver handle.

Returns

- QAPI_OK – Module initialized successfully.
- QAPI_SPIM_ERROR_INVALID_PARAM – Invalid instance or handle parameter.
- QAPI_SPIM_ERROR_OPEN_FAILURE – Failed to Open SPIM.

13.2.2 `qapi_Status_t qapi_SPIM_Power_On (void * spi_Handle)`

Enables the SPI hardware resources for an SPI transaction.

This function enables all resources required for a successful SPI transaction. This includes 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

in	<i>spi_Handle</i>	Driver handle returned by qapi_SPIM_Open() .
----	-------------------	--

Returns

- `QAPI_OK` – SPI master enabled successfully.
- `QAPI_SPIM_ERROR_INVALID_PARAM` – Invalid handle parameter.
- `QAPI_SPIM_ERROR_POWER_ON_FAILURE` – Failure in Power On.

Dependencies

- Before calling this API [qapi_SPIM_Open\(\)](#) API must be called.

13.2.3 `qapi_Status_t qapi_SPIM_Power_Off (void * spi_Handle)`

Disables the SPI hardware resources for an SPI transaction.

This function turns off all resources used by the SPI master. This includes clocks, power resources, and GPIOs. This function should be called to put the SPI master in its lowest possible power state.

Prototype

```
qapi_Status_t qapi_SPIM_Power_Off ( void * spi_Handle )
```

Parameters

in	<code>spi_Handle</code>	Driver handle returned by qapi_SPIM_Open() .
----	-------------------------	--

Returns

- `QAPI_OK` – SPI master disabled successfully.
- `QAPI_SPIM_ERROR_INVALID_PARAM` – Invalid handle parameter.
- `QAPI_SPIM_ERROR_POWER_OFF_FAILURE` – Failure in Power Off.

Dependencies

- Before calling this API [qapi_SPIM_Open\(\)](#) and [qapi_SPIM_Power_On\(\)](#) API must be called.

13.2.4 `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)`

Performs a data transfer over the SPI bus.

This function performs 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

in	<code>spi_Handle</code>	Driver handle returned by qapi_SPIM_Open() .
in	<code>config</code>	Pointer to the SPI configuration structure described by qapi_SPIM_Config_t .
in	<code>desc</code>	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.
in	<code>num_Descriptors</code>	Number of descriptor pointers the client wants to process.
in	<code>c_Fn</code>	Callback function to be invoked when the SPI transfer completes successfully or with an error.
in	<code>c_Ctxt</code>	Pointer to a client object that will be returned as an argument to <code>c_Fn</code> .
in	<code>get_timestamp</code>	Boolean variable to indicate whether a transaction timestamp needs to be provided; this is not supported for the QUPv2 version.

Returns

- QAPI_OK – SPI master was enabled successfully.
- QAPI_SPIM_ERROR_INVALID_PARAM – One or more invalid parameters.
- QAPI_SPIM_ERROR_TRANSFER_CONFIG_FAIL-- Failure in Transfer Configuration.
- QAPI_SPIM_ERROR_TRANSFER_FAILURE – Failure in Transfer.

Dependencies

- Before calling this API [qapi_SPIM_Open\(\)](#) and [qapi_SPIM_Power_On\(\)](#) API must be called.

13.2.5 qapi_Status_t qapi_SPIM_Close (void * spi_handle)

Closes the SPI master.

This function frees all internal data structures and closes the SPI master interface. The handle returned by [qapi_SPIM_Open\(\)](#) is then rendered invalid.

Prototype

```
qapi_Status_t qapi_SPIM_Close ( void * spi_handle )
```

Parameters

in	<i>spi_handle</i>	Driver handle returned by qapi_SPIM_Open() .
----	-------------------	--

Returns

- QAPI_OK – SPI driver closed successfully.
- QAPI_SPIM_ERROR_INVALID_PARAM – One or more invalid parameters.

Dependencies

- Before calling this API [qapi_SPIM_Open\(\)](#) and [qapi_SPIM_Power_On\(\)](#) API must be called.

14 UART APIs

This section describes the UART data types and APIs.

This chapter provides the following QAPIs:

- `qapi_UART_Close`
- `qapi_UART_Open`
- `qapi_UART_Receive`
- `qapi_UART_Transmit`
- `qapi_UART_Power_On`
- `qapi_UART_Power_Off`
- `qapi_UART_Ioctl`

FIBOCOM
Confidential

14.1 SPI Date Types

14.1.1 Data Structure

14.1.1.1 union QAPI_UART_ioctl_Param

IOCTL command ID list of the UART.

Data fields

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.

14.1.1.2 struct qapi_UART_Open_Config_t

Structure for UART configuration.

Data fields

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. Be sure not to violate ISR guidelines. Note: Do not call uart_transmit or uart_receive APIs from this callback.
qapi_UART_ Callback_Fn_t	rx_CB_ISR	Receive callback, called from ISR context. Be sure not to violate ISR guidelines. Note: Do not call uart_transmit or uart_receive APIs from this callback.

14.1.2.1 typedef void *qapi_UART_Handle_t

UART handle that is passed to the client when a UART port is opened.

14.1.2.2 typedef void(* qapi_UART_Callback_Fn_t)(uint32_t num_bytes, void *cb_data)

Transmit and receive operation callback type.

Parameters

in	<i>num_bytes</i>	Number of bytes.
out	<i>cb_data</i>	Pointer to the callback data.

14.1.3.1 enum qapi_UART_Port_Id_e

UART port ID enumeration.

This enumeration is used to specify which port is to be opened during the `uart_open` call.

Enumerator:

QAPI_UART_PORT_001_E	<i>UART</i>	<i>core 01</i>
QAPI_UART_PORT_002_E	<i>UART</i>	<i>core 02</i>
QAPI_UART_PORT_003_E	<i>UART</i>	<i>core 03</i>
QAPI_UART_PORT_004_E	<i>UART</i>	<i>core 04</i>
QAPI_UART_PORT_005_E	<i>UART</i>	<i>core 05</i>
QAPI_UART_PORT_006_E	<i>UART</i>	<i>core 06</i>
QAPI_UART_PORT_007_E	<i>UART</i>	<i>core 07</i>
QAPI_UART_PORT_008_E	<i>UART</i>	<i>core 08</i>
QAPI_UART_PORT_009_E	<i>UART</i>	<i>core 09</i>
QAPI_UART_PORT_0010_E	<i>UART</i>	<i>core 10</i>

14.1.3.2 enum qapi_UART_Bits_Per_Char_e

UART bits per character configuration enumeration.

Enumeration to specify how many UART bits are to be used per character configuration.

Enumerator:

QAPI_UART_5_BITS_PER_CHAR_E	<i>5 bits per character</i>
QAPI_UART_6_BITS_PER_CHAR_E	<i>6 bits per character</i>
QAPI_UART_7_BITS_PER_CHAR_E	<i>7 bits per character</i>
QAPI_UART_8_BITS_PER_CHAR_E	<i>8 bits per character</i>

14.1.3.3 enum qapi_UART_Num_Stop_Bits_e



Enumeration for UART number of stop bits configuration.

Enumerator:

QAPI_UART_1_0_STOP_BITS_E	<i>1.0 stop bits</i>
QAPI_UART_1_5_STOP_BITS_E	<i>1.5 stop bits</i>
QAPI_UART_0_5_STOP_BITS_E	<i>0.5 stop bits</i>
QAPI_UART_2_0_STOP_BITS_E	<i>2.0 stop bits</i>

14.1.3.4 enum qapi_UART_Parity_Mode_e

Enumeration for UART parity mode configuration.

Enumerator:

QAPI_UART_NO_PARITY_E	<i>No parity</i>
QAPI_UART_ODD_PARITY_E	<i>Odd parity</i>
QAPI_UART_EVEN_PARITY_E	<i>Even parity</i>
QAPI_UART_SPACE_PARITY_E	<i>Space parity</i>

14.1.3.5 enum qapi_UART_ioctl_Command_e

IOCTL command ID list of the UART.

Enumerator:

QAPI_SET_FLOW_CTRL_E	<i>Set auto flow control.</i>
QAPI_SET_BAUD_RATE_E	<i>Set baud rate.</i>

14.1.3.6 enum QAPI_Flow_Control_Type

Flow control types for UART.

Enumerator:

QAPI_FCTL_OFF_E *Disable flow control*

QAPI_CTSRFR_AUTO_FCTL_E *Use CTS/RFR flow control with auto RX RFR signal generation.*

FIBOCOM
Confidential

14.2 API Functions

14.2.1 qapi_UART_Close

Closes the UART port.

Releases clock, interrupt, and GPIO handles related to this UART and cancels any pending transfers.

Prototype

```
qapi_Status_t qapi_UART_Close ( qapi_UART_Handle_t handle )
```



Note: Do not call this API from ISR context.

Parameters

in	<i>handle</i>	UART handle provided by qapi_UART_Open() .
----	---------------	--

Returns

- QAPI_OK – Port close was successful.
- QAPI_ERROR – Port close failed.

Dependencies

- Before calling this API [qapi_UART_Open\(\)](#) and [qapi_UART_Power_On\(\)](#) APIs must be called.

14.2.2 qapi_UART_Open

Initializes the UART port.

Opens the UART port and configures 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 )
```



Note: Do not call this API from ISR context.

Parameters

in	<i>handle</i>	UART handle.
in	<i>id</i>	ID of the port to be opened.
in	<i>config</i>	Structure that holds all configuration data.

Returns

- QAPI_OK – Port open was successful.
- QAPI_ERROR – Port open failed.

14.2.3 qapi_UART_Receive

Queues the buffer provided for receiving the data.

This is an asynchronous call. rx_cb_isr is called when the Rx transfer completes. The buffer is owned by the UART driver until rx_cb_isr is called.

There must always be 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 a maximum of two buffers queued at a time.

Prototype

```
qapi_Status_t qapi_UART_Receive ( qapi_UART_Handle_t handle, char * buf, uint32_t buf_Size, void * cb_Data )
```



Note: Do not call this API from ISR context.

Parameters

in	<i>handle</i>	UART handle provided by qapi_UART_Open() .
in	<i>buf</i>	Buffer to be filled with data.
in	<i>buf_Size</i>	Buffer size. Must be ≥4 and a multiple of 4.
in	<i>cb_Data</i>	Callback data to be passed when rx_cb_isr is called during Rx completion.

Returns

- QAPI_OK – Queuing of the receive buffer was successful.
- QAPI_ERROR – Queuing of the receive buffer failed.

- Before calling this API `qapi_UART_Open()` and `qapi_UART_Power_On()` APIs must be called.

FIBOCOM
Confidential

14.2.4 qapi_UART_Transmit

Transmits data from a specified buffer.

This is an asynchronous call. The buffer is queued for Tx, and when transmit is completed, tx_cb_isr is called.

The buffer is owned by the UART driver until tx_cb_isr is called.



Note: Do not call this API from ISR context.

Prototype

```
qapi_Status_t qapi_UART_Transmit ( qapi_UART_Handle_t handle, char *buf, uint32_t bytes_To_Tx, void * cb_Data )
```

Parameters

in	<i>handle</i>	UART handle provided by qapi_UART_Open() .
in	<i>buf</i>	Buffer with data for transmit.
in	<i>bytes_To_Tx</i>	Bytes of data to transmit.
in	<i>cb_Data</i>	Callback data to be passed when tx_cb_isr is called during Tx completion.

Returns

- QAPI_OK – Queuing of the transmit buffer was successful.
- QAPI_ERROR – Queuing of the transmit buffer failed.

Dependencies

- Before calling this API [qapi_UART_Open\(\)](#) and [qapi_UART_Power_On\(\)](#) APIs must be called.

14.2.5 qapi_UART_Power_On

Enables the UART hardware resources for a UART transaction.

This function enables all resources required for a successful UART transaction. This includes 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

in	<i>UART_Handle</i>	Driver handle returned by qapi_UART_Open() .
----	--------------------	--

Returns

- QAPI_OK – UART powered on successfully.
- QAPI_ERROR – UART power on failed.

Dependencies

- Before calling this API [qapi_UART_Open\(\)](#) API must be called.

14.2.6 qapi_UART_Power_Off

Disables the UART hardware resources for a UART transaction.

This function turns off all resources used by the UART master. This includes 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

in	<i>UART_Handle</i>	Driver handle returned by qapi_UART_Open() .
----	--------------------	--

Returns

- QAPI_OK – UART powered off successfully.
- QAPI_ERROR – UART power off failed.

Dependencies

- Before calling this API [qapi_UART_Open\(\)](#) and [qapi_UART_Power_On\(\)](#) APIs must be called.

14.2.7 qapi_UART_ioctl

Controls the UART configurations for a UART transaction.

This function controls the UART configurations apart from the IO operations, which cannot be achieved through standard APIs.

Prototype

```
qapi_Status_t qapi_UART_ioctl ( qapi_UART_Handle_t handle, qapi_UART_ - ioctl_Command_e
ioctl_Command, void * ioctl_Param )
```

Parameters

in	<i>handle</i>	Driver handle returned by qapi_UART_Open() .
in	<i>ioctl_Command</i>	Pass the commands listed with qapi_UART_ioctl_Command_e .
in	<i>ioctl_Param</i>	Pass the argument associated with qapi_UART_ioctl_Command_e .

Returns

- QAPI_OK – UART IOCTL configuration was successful.
- QAPI_ERROR – UART IOCTL configuration failed.

Dependencies

- Before calling this API [qapi_UART_Open\(\)](#) and [qapi_UART_Power_On\(\)](#) APIs must be called.

15 Storage Module

The FTL layer is a wrapper on top of the FLASH FTL layer. the FLASH FTL layer provides APIs for raw NAND read/write/erase access and is responsible for bad block management and logical to physical block conversation.

FIBOCOM
Confidential

15.1 File System Data Types

15.1.1 Data Define

**15.1.1.1 #define QAPI_FTL_ERROR(x) ((qapi_Status_t)(QAPI_ERROR(QAPI_M-
OD_BSP_FTL, x)))**

Utility macro to define QAPI_FTL-specific error types.

15.1.1.2 #define QAPI_FTL_NOT_INIT QAPI_FTL_ERROR(0)

Error returned if FTL APIs are called without calling FTL init first.

15.1.1.3 #define QAPI_FTL_OUT_OF_GOOD_BLOCKS QAPI_FTL_ERROR(1)

Error returned if NAND is out of good blocks.

15.1.1.4 #define QAPI_FTL_ERR_UNKNOWN QAPI_FTL_ERROR(2)

Error returned if registering QAPI handler fails.

15.1.1.5 #define QAPI_FTL_ERR_INVLD_ID QAPI_FTL_ERROR(3)

Error returned if QAPI handler is called with an invalid ID.

15.1.2 Data Structure

15.1.2.1 struct qapi_FS_Stat_Type_s

Statistics type, used in the [qapi_FS_Stat\(\)](#) API.

Data fields

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.
--------	--------	--

15.1.2.2 struct qapi_FS_Statvfs_Type_s

File system information, used in the [qapi_FS_Statvfs\(\)](#) API.

Data fields

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 currently 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.

15.1.2.3 struct qapi_FS_Iter_Entry_s

See the [qapi_FS_Iter_Next\(\)](#) API for information about this structure.

Data fields

Reproduction forbidden without Fibocom Wireless Inc. written authorization - All Rights Reserved.

Type	Parameter	Description
char	file_Path	Name of the directory component.
struct qapi_FS_Stat_Type_s	SBuf	See qapi_FS_Stat_Type_s for information on this structure.
uint32	qapi_FS_D_Stats_Present	Bitmask for the qapi_FS_Stat_Type_s structure that defines which fields are filled when the qapi_FS_Iter_Next() API is called.

15.1.2.4 struct qapi_FTL_info_t

Structure for storing information about a partition.

Data fields

Type	Parameter	Description
uint8_t	device_name	Device name string.
uint32_t	maker_id	Manufacturer ID.
uint32_t	device_id	Device ID.
uint32_t	lpa_count	Number of LPAs in the device.
uint32_t	lpa_size_in_kbytes	LPA size in kB.
uint32_t	erase_block_count	Number of eraseable units in the partition.
uint32_t	erase_block_size_in_kbytes	Erase unit size in kB.

15.1.3 Data Typedef

15.1.3.1 typedef void*qapi_FTL_client_t

15.1.4 Data Enumeration

Flag bits to open a file.

Enumerator:

QAPI_FS_O_RDONLY_E	<i>Open for read only</i>
QAPI_FS_O_WRONLY_E	<i>Open for write only</i>
QAPI_FS_O_RDWR_E	<i>Open for read and write</i>
QAPI_FS_O_CREAT_E	<i>Create the file if it does not exist</i>
QAPI_FS_O_EXCL_E	<i>Fail if the file exists</i>
QAPI_FS_O_TRUNC_E	<i>Truncate the file to zero bytes on successful open</i>
QAPI_FS_O_APPEND_E	<i>All writes will self-seek to the end of the file before writing</i>

Mode bits to open a file.

Enumerator:

QAPI_FS_S_IRUSR_E	<i>Read permission for a user</i>
QAPI_FS_S_IWUSR_E	<i>Write permission for a user</i>
QAPI_FS_S_IXUSR_E	<i>Execute permission for a user</i>
QAPI_FS_S_IRGRP_E	<i>Read permission for a group</i>
QAPI_FS_S_IWGRP_E	<i>Write permission for a group</i>
QAPI_FS_S_IXGRP_E	<i>Execute permission for a group</i>
QAPI_FS_S_IROTH_E	<i>Read permission for other</i>
QAPI_FS_S_IWOTH_E	<i>Write permission for other</i>
QAPI_FS_S_IXOTH_E	<i>Execute permission for other</i>
QAPI_FS_S_ISUID_E	<i>Set UID on execution</i>
QAPI_FS_S_ISGID_E	<i>Set GID on execution</i>

Offset bits to seek a file.

Enumerator:

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*

15.1.4.1 enum qapi_FS_Filename_Rule_e

File name rules.

Enumerator:

QAPI_FS_FILENAME_RULE_8BIT_RELAXED *8-bit relaxed rule.*

QAPI_FS_FILENAME_RULE_FAT *FAT rule.*

QAPI_FS_FILENAME_RULE_FDI *FDI rule.*

15.1.4.2 enum qapi_FS_Filename-Encoding_e

File name encoding schemes.

Enumerator:

QAPI_FS_FILENAME_ENCODING_8BIT *8-bit encoding.*

QAPI_FS_FILENAME_ENCODING_UTF8 *UTF8 encoding.*

15.2 API Functions

15.2.1 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

in	<i>Path</i>	Path of the file that is to be opened.
in	<i>Oflag</i>	<p>Argument that describes how this file is to be opened. It contains one of the following values:</p> <p>QAPI_FS_O_RDONLY_E – Open for read only.</p> <p>QAPI_FS_O_WRONLY_E – Open for write only.</p> <p>QAPI_FS_O_RDWR_E – Open for read and write. In addition, the following flags can be bitwise ORed with this value:</p> <p>QAPI_FS_O_APPEND_E – All writes will self-seek to the end of the file before writing.</p> <p>QAPI_FS_O_CREAT_E – Create the file if it does not exist.</p> <p>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:</p> <p>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.</p> <p>QAPI_FS_O_CREAT_E QAPI_FS_O_EXCL_E – Creates a file but fails if it already exists.</p>

in	Mode	<p>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.</p> <p>One or more of the following permission bits can be ORed as the Mode parameter:</p> <p>QAPI_FS_S_IRUSR_E – Read permission for a user</p> <p>QAPI_FS_S_IWUSR_E – Write permission for a user</p> <p>QAPI_FS_S_IXUSR_E – Execute permission for a user</p> <p>QAPI_FS_S_IRGRP_E – Read permission for a group</p> <p>QAPI_FS_S_IWGRP_E – Write permission for a group</p> <p>QAPI_FS_S_IXGRP_E – Execute permission for a group</p> <p>QAPI_FS_S_IROTH_E – Read permission for other</p> <p>QAPI_FS_S_IWOTH_E – Write permission for other</p> <p>QAPI_FS_S_IXOTH_E – Execute permission for other</p> <p>QAPI_FS_S_ISUID_E – Set UID on execution</p> <p>QAPI_FS_S_ISGID_E – Set GID on execution</p> <p>QAPI_FS_S_ISVTX_E – Sticky bit (hidden attribute in FAT)</p>
out	Fd_ptr	<p>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.</p>

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

QAPI_ERR_EEXIST – Cannot create a file with the path name because another file with the same name exists and an exclusive open is requested or a special (exclusive) file with the same path name exists.

QAPI_ERR_ENOENT – No entry for the path name is found, the file cannot be opened (and a new file is not created because the QAPI_FS_O_CREAT flag was not supplied).



QAPI_ERR_EMFILE – Maximum number of open descriptors is exceeded.

QAPI_ERR_EISDIR – Opening a file with a write flag (QAPI_FS_O_WRONLY or QAPI_FS_O_RDWR) was denied because a directory with the same name exists.

QAPI_ERR_ENOSPC – No space is left on the device.

QAPI_ERR_ENAMETOOLONG – File/directory name exceeded the NAME_MAX limit or the path name exceeded the Path_MAX limit, which is 1024 bytes. The maximum length of a full path name, not including a trailing '\0' character.

QAPI_ERR_ENOMEM – No more dynamic memory is available.

QAPI_ERR_ENODEV – The device does not exist.

QAPI_ERR_ENOTDIR – The file could not be created under a path that is not a directory. Another possibility is an open with the QAPI_FS_O_CREAT flag tried to create a file in the ROM file system.

QAPI_ERR_EINVAL – Invalid parameter or path.



Note:

If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.2 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

in	<i>Path</i>	Path of the file that is to be opened.
in	<i>Oflag</i>	Argument that describes how this file should be opened. See qapi_FS_Open_With_Mode() .
out	<i>Fd_ptr</i>	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.

Returns

See [qapi_FS_Open_With_Mode\(\)](#).



Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.3 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

in	<i>Fd</i>	File descriptor obtained via the qapi_FS_Open() function.
out	<i>Buf</i>	Buffer where the read bytes from the file will be stored.
in	<i>Count</i>	Number of bytes to read from the file.
out	<i>Bytes_Read_Ptr</i>	This is a return from the function with the number of bytes actually read.

Returns

Returns QAPI_OK on success, and -ve error code is returned on failure.



Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.4 qapi_FS_Write

Attempts to write 'Count' bytes of data to the file associated with the specified file descriptor.

The write operation 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

in	<i>Fd</i>	File descriptor obtained via the qapi_FS_Open() function.
in	<i>Buf</i>	Buffer to which the file is to be written.
in	<i>Count</i>	Number of bytes to write to the file.
out	<i>Bytes_Written_Ptr</i>	This is a return from the function with the number of bytes actually written.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.



Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.5 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

in	<i>Fd</i>	File descriptor obtained via the qapi_FS_Open() function.
----	-----------	---

Returns

Returns QAPI_OK on success and -ve erro code is returned on failure.

15.2.6 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

in	<code>Old_Path</code>	Path name before the rename operation.
in	<code>New_Path</code>	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.

Returns

Returns `QAPI_OK` on success and -ve error code is returned on failure.

`QAPI_ERR_EINVAL` – Invalid operation denied. The reasons can be a possible permission access violation or the creation of cycle symbolic links if the rename succeeded.

`QAPI_ERR_EISIR` – The `New_Path` is a directory.

`QAPI_ERR_EXDEV` – A rename operation across different file systems is not permitted.

`QAPI_ERR_ENOTEMPTY` – The `Old_Path` directory is not empty.

15.2.7 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

in	<i>Path</i>	Path of the file whose length is to be truncated.
in	<i>Length</i>	New size of the file. The length is in the range (-4 *1024 *1024 *1024, + 4 *1024 *1024 *1024 -1) bytes.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

QAPI_ERR_EINVAL – Truncation is not possible. Invalid operation or parameter.

QAPI_ERR_ENOENT – A file with the specified path was not found.

QAPI_ERR_ENODEV – The device does not exist.

QAPI_ERR_ENAMETOOLONG – File-name or directory name exceeded the QAPI_FS_NAME_MAX limit, or the path name exceeded the Path_MAX limit. The maximum length of a full path name, not including a trailing '\0' character: Path_MAX = 1024.

QAPI_ERR_EEOF – Truncation is not allowed beyond End of File (EOF) on this file system.

15.2.8 qapi_FS_Seek

Changes the file offset for the opened file descriptor.

Changing the file offset does not modify the file. If the seek proceeds past the end of the file and then writes, the gap will be filled with zero bytes. This gap might not 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

in	<i>Fd</i>	File descriptor obtained via the qapi_FS_Open() API.
in	<i>Offset</i>	New offset of the file.
in	<i>Whence</i>	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.
out	<i>Actual_Offset_Ptr</i>	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.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

QAPI_ERR_EINVAL – Invalid operation.

QAPI_ERR_EBADF – File descriptor was not found.

QAPI_ERR_ESPIPE – Some file descriptors (like pipes and FIFOs) are not seekable.



Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.9 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

in	<i>Path</i>	Path for the directory.
in	<i>Mode</i>	Permission bits of the new directory. See the qapi_FS_Open() API for information on Mode bits.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

QAPI_ERR_ENOENT – No Path was found.

QAPI_ERR_EINVAL – Invalid operation or parameter.

QAPI_ERR_ENOSPC – The operation could not be completed because the device is full.

QAPI_ERR_ENAMETOOLONG – File name or directory name exceeded the NAME_MAX limit, or the path name exceeded the Path_MAX limit. The maximum length of a full path name, not including a trailing '\0' character: Path_MAX = 1024.

15.2.10 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

in	<i>Path</i>	Path of the directory that is to be removed.
----	-------------	--

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

QAPI_ERR_ENOTDIR – The parameter Path is not a directory.

QAPI_ERR_ENOTEMPTY – The directory is not empty.

QAPI_ERR_ETXTBSY – The directory is in use or open.

QAPI_ERR_EINVAL – Invalid parameter.

15.2.11 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

in	<i>Path</i>	File to which the link is to be removed.
----	-------------	--

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

QAPI_ERR_ENOENT – No Path was found.

QAPI_ERR_EPERM – Permission is denied.

QAPI_ERR_ETXTBSY – The file is in use or open.

QAPI_ERR_EINVAL – Invalid parameter.

15.2.12 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

in	<i>Path</i>	File descriptor of the file.
out	<i>SBuf</i>	For information on what is returned in the structure, see struct qapi_FS_Stat_Type_s .

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.



Note:

If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.13 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

in	<i>Fd</i>	Handle to a file obtained using the qapi_FS_Open() API.
out	<i>SBuf</i>	Information is returned in the structure qapi_FS_Stat_Type_s .

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.



Note:

If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.14 qapi_FS_Statvfs

Obtains information about an entire file system.

Gets detailed information about the filesystem specified by the 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

in	<i>Path</i>	Valid path of a file or directory on the mounted file system.
out	<i>SBuf</i>	Information is returned in the structure qapi_FS_Statvfs_Type_s .

Returns

Returns QAPI_OK on success, and -ve error code is returned on failure.



Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

15.2.15 qapi_FS_Iter_Open

Opens a directory and gets a handle to the directory.

This function opens a directory for iteration and gets an opaque handle that can then be passed to [qapi_FS_Iter_Next\(\)](#) to iterate through the entries of the opened directory. This same pointer must be passed to `closedir` to close the iterator.

Prototype

```
qapi_FS_Status_t qapi_FS_Iter_Open ( const char * Path, qapi_FS_Iter_Handle_t * handle )
```

Parameters

in	<i>Path</i>	Valid path of the directory to iterate.
out	<i>handle</i>	Handle provided by the module to the client.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.



Note:

1. Clients should cache the handle. Once lost, it cannot be queried back from the module.
2. If this API is called from user space with MMU enabled then, the parameters should be userspace addresses or else invalid param error will be returned.
3. Support for this QAPI is deprecated.

15.2.16 qapi_FS_Iter_Next

Reads the next entry in the directory using the opened directory iterator.

If an entry is present, the structure parameter is filled with details about the entry. Otherwise, a NULL value is filled.

Prototype

qapi_FS_Status_t qapi_FS_Iter_Next (qapi_FS_Iter_Handle_t Iter_Hdl, struct qapi_FS_Iter_Entry_s * Iter_Entry)



Note:

Any code that uses [qapi_FS_Iter_Next\(\)](#) must be prepared for [qapi_FS_D_Stats_Present\(\)](#) to be zero and call [qapi_FS_Stat\(\)](#) for each entry.

Parameters

in	<i>Iter_Hdl</i>	Handle to directory obtained by the qapi_FS_Iter_Open() API.
out	<i>Iter_Entry</i>	Details about the next entry found is filled in struct qapi_FS_Dirent { char file_Path[QAPI_FS_NAME_MAX+1] struct qapi_FS_Stat_Type_s SBuf uint32 qapi_FS_D_Stats_Present; }

file_Path – Name of the directory component

SBuf – Information about the component; See [qapi_FS_Stat_Type_s](#) for information about this structure

qapi_FS_D_Stats_Present – This is a bitmask for the above structure that defines which fields are filled when this this API is called.

Bitmasks for [qapi_FS_D_Stats_Present](#) are defined as:

QAPI_FS_DIRENT_HAS_ST_DEV = (1 << 1)

QAPI_FS_DIRENT_HAS_ST_INO = (1 << 2)

QAPI_FS_DIRENT_HAS_ST_Mode = (1 << 3)

QAPI_FS_DIRENT_HAS_ST_NLINK = (1 << 4)

QAPI_FS_DIRENT_HAS_ST_SIZE = (1 << 5)

QAPI_FS_DIRENT_HAS_ST_BLKSIZE = (1 << 6)

QAPI_FS_DIRENT_HAS_ST_BLOCKS = (1 << 7)

QAPI_FS_DIRENT_HAS_ST_ATIME = (1 << 8)

QAPI_FS_DIRENT_HAS_ST_MTIME = (1 << 9)

QAPI_FS_DIRENT_HAS_ST_CTIME = (1 << 10)

QAPI_FS_DIRENT_HAS_ST_RDEV = (1 << 11)

QAPI_FS_DIRENT_HAS_ST_UID = (1 << 12)

QAPI_FS_DIRENT_HAS_ST_GID = (1 << 13)

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.



Note:

If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

Support for this QAPI is deprecated.

15.2.17 qapi_FS_Iter_Close

Closes the directory iterator, releasing the iterator for reuse.

Prototype

```
qapi_FS_Status_t qapi_FS_Iter_Close( qapi_FS_Iter_Handle_t Iter_Hdl )
```

Parameters

in	<i>Iter_Hdl</i>	Handle to the directory obtained using the qapi_FS_Iter_Open() API.
----	-----------------	---

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.



Note:

Support for this QAPI is deprecated.

15.2.18 qapi_FS_Last_Error

Returns the last error that occurred in the current task context.

If [qapi_FS_Open\(\)](#) fails, 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 )
```

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

15.2.19 qapi_FTL_Open

Opens an FTL.

This is the first API a client must call before any other call to this module is made.

This API opens the requested partition and returns a handle to that partition. This handle is a void pointer and does not expose any data in and of itself. The handle is to be used with FTL APIs to perform other tasks, e.g., use this handle with [qapi_FTL_Get_info\(\)](#) to get FTL information in the format of [qapi_FTL_info_t](#). As with read and write data functions, this handle must be passed with the correct offset and size.

Prototype

```
qapi_Status_t qapi_FTL_Open ( qapi_FTL_client_t * handle, const uint8_t *part_name )
```



Note:

One handle is returned per partition.



Note:

User space support for this API is deprecated.

Parameters

in	<i>part_name</i>	Name of the partition the client wants to use.
out	<i>handle</i>	Handle that is passed to the client for further use. The client must pass the address of the pointer in which this handle is to be stored. If the return status is QAPI_OK, handle will contain the handle to the partition, which is used for any read or write operation on partition <i>part_name</i> .

Returns

QAPI_ERR_INVALID_PARAM – handle or part_name is NULL, or part_name is invalid.



QAPI_ERROR – An internal failure occurred.

QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.

QAPI_OK – Success.

FIBOCOM
Confidential

15.2.20 qapi_FTL_Close

Closes a partition once the client is done with it.

Prototype

```
qapi_Status_t qapi_FTL_Close ( qapi_FTL_client_t * handle )
```



Note:

User space support for this API is deprecated.

Parameters

in	<i>handle</i>	Handle of the partition to be closed.
----	---------------	---------------------------------------

Returns

QAPI_ERR_INVALID_PARAM – handle is NULL.

QAPI_ERROR – An internal failure occurred.

QAPI_OK – Success.

15.2.21 qapi_FTL_Get_info

Gets partition and client-specific information in a format specified by [qapi_FTL_info_t](#), which can be used to get partition information, such as size.

Prototype

```
qapi_Status_t qapi_FTL_Get_info ( qapi_FTL_client_t handle, qapi_FTL_info_t info )
```



Note:

The total usable partition size in kB = lpa_size_in_kbytes * lpa_count.



Note:

User space support for this API is deprecated.

Parameters

in	<i>handle</i>	Handle returned from qapi_FTL_Open() .
out	<i>info</i>	Pointer to where the information is stored.

Returns

QAPI_ERR_INVALID_PARAM – handle or info is NULL.

QAPI_OK – Success.

15.2.22 qapi_FTL_Read_Ipa

Reads data in multiples of LPA(s) or pages.

Prototype

```
qapi_Status_t qapi_FTL_Read_Ipa ( qapi_FTL_client_t handle, uint32_t lpa, uint32_t lpa_count, uint8_t * buffer )
```



Note:

User space support for this API is deprecated.

Parameters

in	<i>handle</i>	Handle returned from qapi_FTL_Open() .
in	<i>lpa</i>	Logical page address (or page number) from which the data is to be read. The LPA is with respect to the start of the partition.
in	<i>lpa_count</i>	Number of LPAs or pages to read.
out	<i>buffer</i>	Pointer to where the read data is stored. It should be an uncached, physically contiguous DMA-BLE.

Returns

QAPI_ERR_INVALID_PARAM – handle or lpa is NULL.

QAPI_ERROR – An internal failure occurred.

QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.

QAPI_OK – Success.

15.2.23 qapi_FTL_Write_lpa

Writes data in multiples of LPA(s) or pages sequentially.

The number of LPAs in a block = (erase_block_size_in_kbytes/lpa_size_in_kbytes). Data can only be written in an erased block, so before writing in an LPA, the block to which it correspond should be erased by calling [qapi_FTL_Erase_block\(\)](#). For example, if a block has four LPAs, the block is not erased, and the user wants to write in LPA 0, the user must erase the entire block first and then write. Because the entire block is erased, the user does not need to erase before writing in lpa1-lpa3.

Prototype

```
qapi_Status_t qapi_FTL_Write_lpa ( qapi_FTL_client_t handle, uint32_t lpa,uint32_t lpa_count, uint8_t *
buffer )
```



Note:

Only sequential writes are allowed. If the user wants to write in lpa0 after writing in lpa1, the user must erase the entire block. In this case, the data in the entire block is lost. If user wants to write into a previously written LPA, the user must make a back up of the entire block, erase it, and copy in the backed up data. This is the user's responsibility. For example, if the user wants to write in lpa0 after writing in lpa3, the user must follow this sequence:

1. Back up the entire block (if required)
2. Erase the entire block using [qapi_FTL_Erase_block\(\)](#)
3. Write into lpa0
4. Copy lpa1 to lpa3 if a backup was taken before

FTL does not take ownership of a data loss in cases where a sequential write is not followed. Ideally, the

user should erase the whole partition first and then start writing data sequentially.



Note:

User space support for this API is deprecated.

Parameters

in	handle	Handle returned from qapi_FTL_Open() .
in	lpa	Logical page address (or page number) where the data is to be written. The LPA is with respect to the start of the partition

in	<i>lpa_count</i>	Number of LPAs or pages to write.
in	<i>buffer</i>	Pointer to the buffer to which the data is to be written. It should be an uncached, physically contiguous DMA-BLE.

Returns

- QAPI_ERR_INVALID_PARAM – handle or lpa is NULL.
- QAPI_ERROR – An internal failure occurred.
- QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.
- QAPI_OK – Success.

15.2.24 qapi_FTL_Erase_block

Erases a block.

The block size is defined by `erase_block_size_in_kbytes`. The number of LPAs in a block = $(\text{erase_block_size_in_kbytes} / \text{lpa_size_in_kbytes})$. Data can only be written in an erased block, so before writing in an LPA, the block to which it corresponds to must be erased with this API.

Prototype

```
qapi_Status_t qapi_FTL_Erase_block ( qapi_FTL_client_t handle, uint32_t erase_block, uint32_t
erase_block_count )
```

Parameters

in	<code>handle</code>	Handle returned from qapi_FTL_Open() .
in	<code>erase_block</code>	Start erase block.
in	<code>erase_block_count</code>	Number of blocks to be erased from Flash starting at <code>erase_block</code> .

Returns

QAPI_ERR_INVALID_PARAM – handle is NULL.

QAPI_ERROR – An internal failure occurred.

QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.

16 Timer

This chapter provides the following APIs for timer services. This timer service is different than the RTOS timer service

This chapter provides the following QAPIs:

- [qapi_time_get](#)
- [qapi_Timer_Def](#)
- [qapi_Timer_Set](#)
- [qapi_Timer_Get_Timer_Info](#)
- [qapi_Timer_Sleep](#)
- [qapi_Timer_Undef](#)
- [qapi_Timer_Stop](#)

16.1 Timer Data Types

16.1.1 Data Structure

16.1.1.1 struct qapi_TIMER_define_attr_t

Timer define attribute type.

This type is used to specify parameters when defining a timer.

sigs_func_ptr will depend on the value of qapi_TIMER_notify_t.

qapi_TIMER_notify_t == QAPI_TIMER_NO_NOTIFY_TYPE,

sigs_func_ptr = Don't care

qapi_TIMER_notify_t == QAPI_TIMER_NATIVE_OS_SIGNAL_TYPE,

sigs_func_ptr = qurt signal object

qapi_TIMER_notify_t == QAPI_TIMER_FUNC1_CB_TYPE,

sigs_func_ptr == specify a callback of type qapi_TIMER_cb_t

Data fields

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.

16.1.1.2 struct qapi_TIMER_get_cbinfo_t

This type gets the callback information of a user space expired timer.

Reproduction forbidden without Fibocom Wireless Inc. written authorization - All Rights Reserved.

data = Specify the callback data for func_ptr,
func_ptr = function pointer needs to be invoked.

Data fields

Type	Parameter	Description
void *	func_ptr	Specify the callback function.
uint32_t	data	Specify the callback data.

16.1.1.3 struct qapi_TIMER_set_attr_t

Type used to specify parameters when starting a timer.

Data fields

Type	Parameter	Description
uint64_t	time	Timer duration.
uint64_t	reload	Reload duration.
qapi_TIMER_ unit_type	unit	Specify units for timer duration.

16.1.1.4 struct qapi_TIMER_get_info_attr_t

Type used to get information for a given timer.

Data fields

Type	Parameter	Description
qapi_TIMER_ info_type	timer_info	Timer information type.
qapi_TIMER_ unit	unit	Specify units to use for return.

unit_type		
-----------	--	--

16.1.1.5 struct qapi_time_julian_type

Time in Julian format.

Data fields

Type	Parameter	Description
uint16_t	year	Year (1980 through 2100).
uint16_t	month	Month of the year (1 through 12).
uint16_t	day	Day of the month (1 through 31).
uint16_t	hour	Hour of the day (0 through 23).
uint16_t	minute	Minute of the hour (0 through 59).
uint16_t	second	Second of the minute (0 through 59).
uint16_t	day_of_week	Day of the week (0 through 6 or Monday through Sunday).

16.1.1.6 struct node

Timer node.

User space timers are stored in a linked list. Each node of the linked list is of this type.

Data fields

Type	Parameter	Description
uint16_t	index	
struct node *	next	

16.1.1.7 union qapi_time_get_t

Reproduction forbidden without Fibocom Wireless Inc. written authorization - All Rights Reserved.



Used to specify parameters when getting the time.

Pointers depend on the value of `qapi_time_unit_type`

```
qapi_time_unit_type == QAPI_TIME_STAMP,  
time_ts = Of type qapi_time_type
```

```
qapi_time_unit_type == QAPI_TIME_MSECS,  
time_msecs = Of type uint64_t
```

```
qapi_time_unit_type == QAPI_TIME_SECS,  
time_secs = Of type uint64_t
```

```
qapi_time_unit_type == QAPI_TIME_JULIAN,  
time_julian = Of type qapi_time_julian_type
```

Data fields

Type	Parameter	Description
qapi_time_type	<code>time_ts</code>	Specify the <code>qapi_time_type</code> variable pointer.
<code>uint64_t</code>	<code>time_msecs</code>	Variable for getting time in msec.
<code>uint64_t</code>	<code>time_secs</code>	Variable for getting time in sec.
qapi_time_julian_type	<code>time_julian</code>	Variable for getting time in Julian.

16.1.2 Data Typedef

16.1.2.1 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.

16.1.2.2 typedef void (*qapi_TIMER_cb_t)(uint32_t data)

Timer callback type.

Timer callbacks should adhere to this signature.

16.1.2.3 typedef unsigned long qapi_qword[2]

Time type.

Native timestamp type.

16.1.2.4 typedef qapi_qword qapi_time_type

Time type.

Native timestamp type. Same as qapi_qword.

16.1.3 Data Enumeration

16.1.3.1 enum qapi_TIMER_notify_t

Timer notification type.

Enumeration of the notifications available on timer expiry.

Enumerator:

QAPI_TIMER_NO_NOTIFY_TYPE	<i>No notification</i>
QAPI_TIMER_NATIVE_OS_SIGNAL_TYPE	<i>Signal an object</i>
QAPI_TIMER_FUNC1_CB_TYPE	<i>Call back a function</i>

16.1.3.2 enum qapi_TIMER_unit_type

Timer unit type.

Enumeration of the units in which timer duration can be specified.

Enumerator:

QAPI_TIMER_UNIT_TICK	<i>Return time in ticks</i>
QAPI_TIMER_UNIT_USEC	<i>Return time in microseconds</i>
QAPI_TIMER_UNIT_MSEC	<i>Return time in milliseconds</i>
QAPI_TIMER_UNIT_SEC	<i>Return time in seconds</i>
QAPI_TIMER_UNIT_MIN	<i>Return time in minutes</i>
QAPI_TIMER_UNIT_HOUR	<i>Return time in hours</i>

16.1.3.3 enum qapi_TIMER_info_type

Timer information type.



Enumeration of the types of information that can be obtained for a timer.

Enumerator:

QAPI_TIMER_TIMER_INFO_ABS_EXPIRY Return the timetick of timer expiry in native ticks

QAPI_TIMER_TIMER_INFO_TIMER_DURATION Return the total duration of the timer in specified units

QAPI_TIMER_TIMER_INFO_TIMER_REMAINING Return the remaining duration of the timer in specified units

16.1.3.4 enum qapi_time_unit_type

Time unit type.

Enumeration of the types of time that can be obtained from time get QAPI.

Enumerator:

QAPI_TIME_STAMP Return the time in timestamp format

QAPI_TIME_MSECS Return the time in millisecond format

QAPI_TIME_SECS Return the time in second format

QAPI_TIME_JULIAN Return the time in Julian calendar format

16.2 API Functions

16.2.1 qapi_time_get

Gets the time in the specified format.

Prototype

```
qapi_Status_t qapi_time_get ( qapi_time_unit_type time_get_unit, qapi_time_get_t * time )
```

Parameters

in	<i>time_get_unit</i>	Unit in which to get the time.
in	<i>time</i>	Pointer to the qapi_time_get_t variable.

Returns

QAPI_OK on success, an error code on failure.

16.2.2qapi_Timer_Def

Allocates internal memory in the timer module. The internal memory is then formatted with parameters provided in the timer_def_attr variable. The allocated timer pointer is kept in a table. A unique integer is returned to the client as timer_handle. 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

in	<i>timer_handle</i>	Handle to the timer.
in	<i>timer_attr</i>	Attributes for defining the timer.

Returns

QAPI_OK on success, an error code on failure.

Side effects

Calling this API causes memory allocation. Therefore, whenever the timer usage is done and not required, [qapi_Timer_Undef\(\)](#) must be called to release the memory, otherwise it will cause a memory leak.

16.2.3qapi_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 its expiry.

Prototype

```
qapi_Status_t qapi_Timer_Set ( qapi_TIMER_handle_t timer_handle,qapi_TIMER_set_attr_t *
timer_attr )
```

Parameters

in	<i>timer_handle</i>	Handle to the timer.
in	<i>timer_attr</i>	Attributes for setting the timer.

Returns

QAPI_OK on success, an error code on failure.

Dependencies

The [qapi_Timer_Def\(\)](#) API should be called for the timer before calling qapi_Timer_Set function.

16.2.4qapi_Timer_Get_Timer_Info

Gets 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

in	<i>timer_handle</i>	Handle to the timer.
out	<i>timer_info</i>	Type of information needed from the timer.
out	<i>data</i>	Returned timer information.

Returns

QAPI_OK on success, an error code is returned on failure.

16.2.5qapi_Timer_Sleep

Timed wait. Blocks a thread for a specified time. Parameters

Prototype

```
qapi_Status_t qapi_Timer_Sleep ( uint64_t timeout, qapi_TIMER_unit_type unit, qbool_t non_deferrable )
```

Parameters

in	<i>timeout</i>	Specify the duration to block the thread.
in	<i>unit</i>	Specify the units of the duration.
in	<i>non_deferrable</i>	<p>TRUE = processor (if in deep sleep or power collapse) will be awakened on timeout.</p> <p>FALSE = processor will not be awakened from deep sleep or power collapse on timeout.</p> <p>Whenever the processor wakes up due to some other reason after timeout, the thread will be unblocked.</p>

Returns

QAPI_OK on success, an error code on failure.

16.2.6qapi_Timer_Undef

Undefines the timer.

This function must be called whenever timer usage is done. Calling this function releases the internal timer memory that was allocated when the timer was defined, makes the corresponding entry in the qapi timer table NULL, and adds a node in the list of free indices.

Prototype

```
qapi_Status_t qapi_Timer_Undef ( qapi_TIMER_handle_t timer_handle )
```

Parameters

in	<i>timer_handle</i>	Timer handle for which to undefine the timer.
----	---------------------	---

Returns

QAPI_OK on success, an error code on failure.

16.2.7qapi_Timer_Stop

Stops the timer.



Note: This function does not deallocate the memory that was allocated when the timer was defined.

Prototype

```
qapi_Status_t qapi_Timer_Stop ( qapi_TIMER_handle_t timer_handle )
```

Parameters

in	<i>timer_handle</i>	Timer handle for which to stop the timer.
----	---------------------	---

Returns

QAPI_OK on success, an error code on failure.