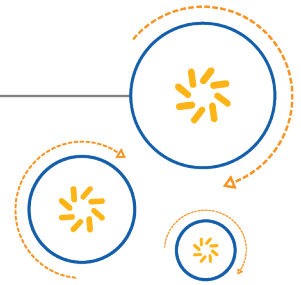




Qualcomm Technologies, Inc.



# Qualcomm Application Programming Interface for MDM9206 ThreadX OS

## Interface Specification

80-P8101-14 A

May 5, 2017

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

**Confidential and Proprietary - Qualcomm Technologies, Inc.**

**NO PUBLIC DISCLOSURE PERMITTED:** Please report postings of this document on public servers or websites to:

[DocCtrlAgent@qualcomm.com](mailto:DocCtrlAgent@qualcomm.com).

**Restricted Distribution.** Not to be distributed to anyone who is not an employee of either Qualcomm Technologies, Inc. or its affiliated companies without the express approval of Qualcomm Configuration Management.

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer ("export") laws. Diversion contrary to U.S. and international law is strictly prohibited.

Qualcomm Technologies, Inc.  
5775 Morehouse Drive  
San Diego, CA 92121  
U.S.A.

© 2017 Qualcomm Technologies, Inc. All rights reserved.

## Revision History

Revision	Date	Description
A	May 2017	Initial release

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

# Contents

---

<b>1</b>	<b>Introduction</b>	<b>25</b>
1.1	Purpose	25
1.2	Conventions	25
1.3	Technical Assistance	25
<b>2</b>	<b>Data Call Functional Overview</b>	<b>26</b>
2.1	Data call architecture in the ThreadX OS	26
<b>3</b>	<b>DSS Net Control APIs</b>	<b>27</b>
3.1	DSS Netctrl Macros, Data Structures, and Enumerations	28
3.1.1	Define Documentation	29
3.1.1.1	QAPI_DSS_RADIO_TECH_UNKNOWN	29
3.1.1.2	QAPI_DSS_RADIO_TECH_MIN	29
3.1.1.3	QAPI_DSS_RADIO_TECH_UMTS	29
3.1.1.4	QAPI_DSS_RADIO_TECH_CDMA	29
3.1.1.5	QAPI_DSS_RADIO_TECH_1X	29
3.1.1.6	QAPI_DSS_RADIO_TECH_DO	29
3.1.1.7	QAPI_DSS_RADIO_TECH_LTE	29
3.1.1.8	QAPI_DSS_RADIO_TECH_TDSCDMA	29
3.1.1.9	QAPI_DSS_CALL_INFO_USERNAME_MAX_LEN	29
3.1.1.10	QAPI_DSS_CALL_INFO_PASSWORD_MAX_LEN	29
3.1.1.11	QAPI_DSS_CALL_INFO_APN_MAX_LEN	29
3.1.1.12	QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN	30
3.1.1.13	QAPI_DSS_SUCCESS	30
3.1.1.14	QAPI_DSS_ERROR	30
3.1.1.15	QAPI_DSS_IP_VERSION_4	30
3.1.1.16	QAPI_DSS_IP_VERSION_6	30
3.1.1.17	QAPI_DSS_IP_VERSION_4_6	30
3.1.1.18	QAPI_DSS_MODE_GENERAL	30
3.1.2	Data Structure Documentation	30
3.1.2.1	struct qapi_DSS_CE_Reason_s	30
3.1.2.2	struct qapi_DSS_Call_Param_Value_s	30
3.1.2.3	struct qapi_DSS_Addr_s	31
3.1.2.4	union qapi_DSS_Addr_s::qapi_dss_ip_address_u	31
3.1.2.5	struct qapi_DSS_Addr_Info_s	31
3.1.2.6	struct qapi_DSS_Data_Pkt_Stats_s	31
3.1.3	Typedef Documentation	32
3.1.3.1	qapi_DSS_Net_Ev_CB_t	32
3.1.4	Enumeration Type Documentation	32

3.1.4.1	qapi_DSS_Auth_Pref_e	32
3.1.4.2	qapi_DSS_CE_Reason_Type_e	32
3.1.4.3	qapi_DSS_Call_Info_Enum_e	33
3.1.4.4	qapi_DSS_Net_Evt_e	33
3.1.4.5	qapi_DSS_IP_Family_e	33
3.1.4.6	qapi_DSS_Data_Bearer_Tech_e	33
3.1.4.7	qapi_DSS_Call_Tech_Type_e	34
3.2	Initialize the DSS Netctrl Library	35
3.2.1	Function Documentation	35
3.2.1.1	qapi_DSS_Init	35
3.3	Release the DSS Netctrl Library	36
3.3.1	Function Documentation	36
3.3.1.1	qapi_DSS_Release	36
3.4	Get the Data Service Handle	37
3.4.1	Function Documentation	37
3.4.1.1	qapi_DSS_Get_Data_Srvc_Hndl	37
3.5	Release the Data Service Handle	38
3.5.1	Function Documentation	38
3.5.1.1	qapi_DSS_Rel_Data_Srvc_Hndl	38
3.6	Set the Data Call Parameter	39
3.6.1	Function Documentation	39
3.6.1.1	qapi_DSS_Set_Data_Call_Param	39
3.7	Start a Data Call	40
3.7.1	Function Documentation	40
3.7.1.1	qapi_DSS_Start_Data_Call	40
3.8	Stop a Data Call	41
3.8.1	Function Documentation	41
3.8.1.1	qapi_DSS_Stop_Data_Call	41
3.9	Get Packet Data Transfer Statistics	42
3.9.1	Function Documentation	42
3.9.1.1	qapi_DSS_Get_Pkt_Stats	42
3.10	Reset Packet Data Transfer Statistics	43
3.10.1	Function Documentation	43
3.10.1.1	qapi_DSS_Reset_Pkt_Stats	43
3.11	Get the Data Call End Reason	44
3.11.1	Function Documentation	44
3.11.1.1	qapi_DSS_Get_Call_End_Reason	44
3.12	Get the Data Call Technology	45
3.12.1	Function Documentation	45
3.12.1.1	qapi_DSS_Get_Call_Tech	45
3.13	Get the Data Bearer Technology	46
3.13.1	Function Documentation	46
3.13.1.1	qapi_DSS_Get_Current_Data_Bearer_Tech	46
3.14	Get the Device Name	47
3.14.1	Function Documentation	47
3.14.1.1	qapi_DSS_Get_Device_Name	47
3.15	Get the QMI Port Name	48
3.15.1	Function Documentation	48
3.15.1.1	qapi_DSS_Get_Qmi_Port_Name	48
3.16	Get the IP Address Count	49

3.16.1	Function Documentation	49
3.16.1.1	qapi_DSS_Get_IP_Addr_Count	49
3.17	Get the IP Address Information	50
3.17.1	Function Documentation	50
3.17.1.1	qapi_DSS_Get_IP_Addr	50
3.18	Get the IP Address Information Structure	51
3.18.1	Function Documentation	51
3.18.1.1	qapi_DSS_Get_IP_Addr_Per_Family	51
3.19	Get the Link MTU Information	52
3.19.1	Function Documentation	52
3.19.1.1	qapi_DSS_Get_Link_Mtu	52
<b>4</b>	<b>QAPI Networking Socket</b>	<b>53</b>
4.1	QAPI Socket Macros and Data Structures	55
4.1.1	Define Documentation	57
4.1.1.1	AF_UNSPEC	57
4.1.1.2	AF_INET	57
4.1.1.3	AF_INET6	57
4.1.1.4	AF_INET_DUAL46	57
4.1.1.5	SOCK_STREAM	58
4.1.1.6	SOCK_DGRAM	58
4.1.1.7	SOCK_RAW	58
4.1.1.8	ENOBUFS	58
4.1.1.9	ETIMEDOUT	58
4.1.1.10	EISCONN	58
4.1.1.11	EOPNOTSUPP	58
4.1.1.12	ECONNABORTED	58
4.1.1.13	EWOULDBLOCK	58
4.1.1.14	ECONNREFUSED	58
4.1.1.15	ECONNRESET	58
4.1.1.16	ENOTCONN	58
4.1.1.17	EBADF	59
4.1.1.18	EALREADY	59
4.1.1.19	EINVAL	59
4.1.1.20	EMSGSIZE	59
4.1.1.21	EPIPE	59
4.1.1.22	EDESTADDRREQ	59
4.1.1.23	ESHUTDOWN	59
4.1.1.24	ENOPROTOOPT	59
4.1.1.25	EHAVEOOB	59
4.1.1.26	ENOMEM	59
4.1.1.27	EADDRNOTAVAIL	59
4.1.1.28	EADDRINUSE	59
4.1.1.29	EAFNOSUPPORT	60
4.1.1.30	EINPROGRESS	60
4.1.1.31	ELOWER	60
4.1.1.32	ENOTSOCK	60
4.1.1.33	EIEIO	60
4.1.1.34	ETOOMANYREFS	60
4.1.1.35	EFAULT	60

4.1.1.36	ENETUNREACH	60
4.1.1.37	SOL_SOCKET	60
4.1.1.38	SOL_SOCKET	60
4.1.1.39	SO_ACCEPTCONN	60
4.1.1.40	SO_REUSEADDR	60
4.1.1.41	SO_KEEPAIVE	61
4.1.1.42	SO_DONTROUTE	61
4.1.1.43	SO_BROADCAST	61
4.1.1.44	SO_USELOOPBACK	61
4.1.1.45	SO_LINGER	61
4.1.1.46	SO_OOINLINE	61
4.1.1.47	SO_TCPSACK	61
4.1.1.48	SO_WINSIZE	61
4.1.1.49	SO_TIMESTAMP	61
4.1.1.50	SO_BIGCWND	61
4.1.1.51	SO_HDRINCL	61
4.1.1.52	SO_NOSLOWSTART	61
4.1.1.53	SO_FULLMSS	62
4.1.1.54	SO_SNDTIMEO	62
4.1.1.55	SO_RCVTIMEO	62
4.1.1.56	SO_ERROR	62
4.1.1.57	SO_RXDATA	62
4.1.1.58	SO_TXDATA	62
4.1.1.59	SO_MYADDR	62
4.1.1.60	SO_NBIO	62
4.1.1.61	SO_BIO	62
4.1.1.62	SO_NONBLOCK	62
4.1.1.63	SO_CALLBACK	62
4.1.1.64	SO_UDPCALLBACK	62
4.1.1.65	IPPROTO_IP	63
4.1.1.66	IP_HDRINCL	63
4.1.1.67	IP_MULTICAST_IF	63
4.1.1.68	IP_MULTICAST_TTL	63
4.1.1.69	IP_MULTICAST_LOOP	63
4.1.1.70	IP_ADD_MEMBERSHIP	63
4.1.1.71	IP_DROP_MEMBERSHIP	63
4.1.1.72	IPV6_MULTICAST_IF	63
4.1.1.73	IPV6_MULTICAST_HOPS	63
4.1.1.74	IPV6_MULTICAST_LOOP	63
4.1.1.75	IPV6_JOIN_GROUP	63
4.1.1.76	IPV6_LEAVE_GROUP	63
4.1.1.77	IP_OPTIONS	64
4.1.1.78	IP_TOS	64
4.1.1.79	IP_TTL_OPT	64
4.1.1.80	IPV6_SCOPEID	64
4.1.1.81	IPV6_UNICAST_HOPS	64
4.1.1.82	IPV6_TCLASS	64
4.1.1.83	MSG_OOB	64
4.1.1.84	MSG_PEEK	64
4.1.1.85	MSG_DONTROUTE	64

4.1.1.86	MSG_DONTWAIT	64
4.1.1.87	MSG_ZEROCOPYSEND	64
4.1.1.88	QAPI_NET_WAIT_FOREVER	64
4.1.1.89	FD_ZERO	65
4.1.1.90	FD_CLR	65
4.1.1.91	FD_SET	65
4.1.1.92	FD_ISSET	65
4.1.2	Data Structure Documentation	65
4.1.2.1	struct in_addr	65
4.1.2.2	struct sockaddr_in	65
4.1.2.3	struct in6_addr	65
4.1.2.4	struct ip46addr_n	66
4.1.2.5	union ip46addr_n.a	66
4.1.2.6	union ip46addr_n.g	66
4.1.2.7	struct sockaddr_in6	66
4.1.2.8	struct ip46addr	67
4.1.2.9	union ip46addr.a	67
4.1.2.10	struct sockaddr	67
4.1.2.11	struct fd_set	67
4.2	Create a Socket	68
4.2.1	Function Documentation	68
4.2.1.1	qapi_socket	68
4.3	Bind a Socket	69
4.3.1	Function Documentation	69
4.3.1.1	qapi_bind	69
4.4	Make a Socket Passive	70
4.4.1	Function Documentation	70
4.4.1.1	qapi_listen	70
4.5	Accept a Socket Connection Request	71
4.5.1	Function Documentation	71
4.5.1.1	qapi_accept	71
4.6	Connect to a Socket	72
4.6.1	Function Documentation	72
4.6.1.1	qapi_connect	72
4.7	Set Socket Options	73
4.7.1	Function Documentation	73
4.7.1.1	qapi_setsockopt	73
4.8	Get Socket Options	74
4.8.1	Function Documentation	74
4.8.1.1	qapi_getsockopt	74
4.9	Close a Socket	75
4.9.1	Function Documentation	75
4.9.1.1	qapi_socketclose	75
4.10	Get a Socket Error Code	76
4.10.1	Function Documentation	76
4.10.1.1	qapi_errno	76
4.11	Receive a Message from a Socket	77
4.11.1	Function Documentation	77
4.11.1.1	qapi_recvfrom	77
4.12	Receive a Message from a Connected Socket	78

4.12.1	Function Documentation	78
4.12.1.1	qapi_rcv	78
4.13	Send a Message on a Socket	79
4.13.1	Function Documentation	79
4.13.1.1	qapi_sendto	79
4.14	Send a Message on a Connected Socket	80
4.14.1	Function Documentation	80
4.14.1.1	qapi_send	80
4.15	Select a Socket	81
4.15.1	Function Documentation	81
4.15.1.1	qapi_select	81
4.16	Initialize a Socket	82
4.16.1	Function Documentation	82
4.16.1.1	qapi_fd_zero	82
4.17	Clear a Socket from a Socket Set	83
4.17.1	Function Documentation	83
4.17.1.1	qapi_fd_clr	83
4.18	Add a Socket to a Socket Set	84
4.18.1	Function Documentation	84
4.18.1.1	qapi_fd_set	84
4.19	Check Whether a Socket is in a Socket Set	85
4.19.1	Function Documentation	85
4.19.1.1	qapi_fd_isset	85
4.20	Get the Address of a Connected Peer	86
4.20.1	Function Documentation	86
4.20.1.1	qapi_getpeername	86
4.21	Get the Address to Which the Socket is Bound	87
4.21.1	Function Documentation	87
4.21.1.1	qapi_getsockname	87
<b>5</b>	<b>QAPI Network Security APIs</b>	<b>88</b>
5.1	QAPI SSL Data Types	89
5.1.1	Define Documentation	89
5.1.1.1	QAPI_NET_SSL_MAX_CERT_NAME_LEN	89
5.1.1.2	QAPI_NET_SSL_MAX_NUM_CERTS	89
5.1.1.3	QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH	89
5.1.1.4	QAPI_NET_SSL_INVALID_HANDLE	89
5.1.1.5	QAPI_NET_SSL_PROTOCOL_UNKNOWN	89
5.1.1.6	QAPI_NET_SSL_PROTOCOL_TLS_1_0	89
5.1.1.7	QAPI_NET_SSL_PROTOCOL_TLS_1_1	89
5.1.1.8	QAPI_NET_SSL_PROTOCOL_TLS_1_2	89
5.1.1.9	QAPI_NET_SSL_PROTOCOL_DTLS_1_0	89
5.1.1.10	QAPI_NET_SSL_PROTOCOL_DTLS_1_2	89
5.1.1.11	QAPI_NET_TLS_PSK_WITH_RC4_128_SHA	90
5.1.1.12	QAPI_NET_TLS_PSK_WITH_3DES_EDE_CBC_SHA	90
5.1.1.13	QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA	90
5.1.1.14	QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA	90
5.1.1.15	QAPI_NET_TLS_PSK_WITH_AES_128_GCM_SHA256	90
5.1.1.16	QAPI_NET_TLS_PSK_WITH_AES_256_GCM_SHA384	90
5.1.1.17	QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA256	90



5.1.1.18	QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA384	90
5.1.1.19	QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA	90
5.1.1.20	QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA	90
5.1.1.21	QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA	90
5.1.1.22	QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA	90
5.1.1.23	QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA256	91
5.1.1.24	QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA256	91
5.1.1.25	QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	91
5.1.1.26	QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	91
5.1.1.27	QAPI_NET_TLS_RSA_WITH_AES_128_GCM_SHA256	91
5.1.1.28	QAPI_NET_TLS_RSA_WITH_AES_256_GCM_SHA384	91
5.1.1.29	QAPI_NET_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	91
5.1.1.30	QAPI_NET_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	91
5.1.1.31	QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	91
5.1.1.32	QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	91
5.1.1.33	QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	91
5.1.1.34	QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	91
5.1.1.35	QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	92
5.1.1.36	QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	92
5.1.1.37	QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	92
5.1.1.38	QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	92
5.1.1.39	QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	92
5.1.1.40	QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	92
5.1.1.41	QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	92
5.1.1.42	QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	92
5.1.1.43	QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	92
5.1.1.44	QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	92
5.1.1.45	QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	92
5.1.1.46	QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	93
5.1.1.47	QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	93
5.1.1.48	QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	93
5.1.1.49	QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	93
5.1.1.50	QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	93
5.1.1.51	QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	93
5.1.1.52	QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	93
5.1.1.53	QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	93
5.1.1.54	QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	93
5.1.1.55	QAPI_NET_TLS_RSA_WITH_AES_128_CCM	93
5.1.1.56	QAPI_NET_TLS_RSA_WITH_AES_256_CCM	93
5.1.1.57	QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM	94
5.1.1.58	QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM	94
5.1.1.59	QAPI_NET_TLS_RSA_WITH_AES_128_CCM_8	94
5.1.1.60	QAPI_NET_TLS_RSA_WITH_AES_256_CCM_8	94
5.1.1.61	QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM_8	94
5.1.1.62	QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM_8	94
5.1.1.63	QAPI_NET_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	94
5.1.1.64	QAPI_NET_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	94
5.1.1.65	QAPI_NET_TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	94

5.1.1.66	QAPI_NET_SSL_MAX_CA_LIST	94
5.1.2	Data Structure Documentation	94
5.1.2.1	struct __qapi_Net_SSL_Verify_Policy_s	94
5.1.2.2	struct __qapi_Net_SSL_Config_s	95
5.1.2.3	struct __qapi_Net_SSL_Cert_List_s	95
5.1.2.4	struct __qapi_Net_SSL_CERT_s	95
5.1.2.5	struct __qapi_NET_SSL_CA_Info_s	96
5.1.2.6	struct __qapi_Net_SSL_CA_List_s	96
5.1.2.7	struct __qapi_Net_SSL_PSK_Table_s	96
5.1.2.8	struct __qapi_Net_SSL_Cert_Info_s	96
5.1.2.9	union __qapi_Net_SSL_Cert_Info_s.info	97
5.1.3	Enumeration Type Documentation	97
5.1.3.1	qapi_Net_SSL_Role_t	97
5.1.3.2	qapi_Net_SSL_Protocol_t	97
5.1.3.3	qapi_Net_SSL_Cert_Type_t	97
5.2	QAPI SSL Typedefs	98
5.2.1	Typedef Documentation	98
5.2.1.1	qapi_Net_SSL_Obj_Hdl_t	98
5.2.1.2	qapi_Net_SSL_Con_Hdl_t	98
5.2.1.3	qapi_Net_SSL_Cert_t	98
5.2.1.4	qapi_Net_SSL_CAList_t	98
5.2.1.5	qapi_Net_SSL_PSKTable_t	98
5.3	Create an SSL Object	99
5.3.1	Function Documentation	99
5.3.1.1	qapi_Net_SSL_Obj_New	99
5.4	Create an SSL Connection Handle	100
5.4.1	Function Documentation	100
5.4.1.1	qapi_Net_SSL_Con_New	100
5.5	Configure an SSL Connection	101
5.5.1	Function Documentation	101
5.5.1.1	qapi_Net_SSL_Configure	101
5.6	Delete an SSL Certificate	103
5.6.1	Function Documentation	103
5.6.1.1	qapi_Net_SSL_Cert_delete	103
5.7	Store an SSL Certificate	104
5.7.1	Function Documentation	104
5.7.1.1	qapi_Net_SSL_Cert_Store	104
5.8	Convert and Store an SSL Certificate	105
5.8.1	Function Documentation	105
5.8.1.1	qapi_Net_SSL_Cert_Convert_And_Store	105
5.9	Load an SSL Certificate	106
5.9.1	Function Documentation	106
5.9.1.1	qapi_Net_SSL_Cert_Load	106
5.10	Get a List of SSL Certificates	107
5.10.1	Function Documentation	107
5.10.1.1	qapi_Net_SSL_Cert_List	107
5.11	Attach a Socket Descriptor to the SSL Connection	108
5.11.1	Function Documentation	108
5.11.1.1	qapi_Net_SSL_Fd_Set	108
5.12	Accept an SSL Connection From the Client	109

5.12.1	Function Documentation	109
5.12.1.1	qapi_Net_SSL_Accept	109
5.13	Initiate an SSL Handshake	110
5.13.1	Function Documentation	110
5.13.1.1	qapi_Net_SSL_Connect	110
5.14	Close an SSL Connection	111
5.14.1	Function Documentation	111
5.14.1.1	qapi_Net_SSL_Shutdown	111
5.15	Free an SSL Object Handle	112
5.15.1	Function Documentation	112
5.15.1.1	qapi_Net_SSL_Obj_Free	112
5.16	Read SSL Data	113
5.16.1	Function Documentation	113
5.16.1.1	qapi_Net_SSL_Read	113
5.17	Write SSL Data	114
5.17.1	Function Documentation	114
5.17.1.1	qapi_Net_SSL_Write	114
<b>6</b>	<b>QAPI Networking Services</b>	<b>115</b>
6.1	Networking Services Macros, Data Types, and Enumerations	116
6.1.1	Define Documentation	116
6.1.1.1	QAPI_IPV4_IS_MULTICAST	116
6.1.1.2	IF_NAMELEN	116
6.1.1.3	QAPI_NET_IPV4_MAX_ROUTES	116
6.1.1.4	QAPI_IS_IPV6_LINK_LOCAL	116
6.1.1.5	QAPI_IS_IPV6_MULTICAST	117
6.1.1.6	QAPI_NET_IPV6_MAX_ROUTES	117
6.1.1.7	QAPI_NET_IFNAME_LEN	117
6.1.2	Data Structure Documentation	117
6.1.2.1	struct qapi_Net_Ping_V4_s	117
6.1.2.2	struct qapi_Net_IPv4_Route_t	117
6.1.2.3	struct qapi_Net_IPv4_Route_List_t	118
6.1.2.4	struct qapi_Net_Ping_V6_s	118
6.1.2.5	struct qapi_Net_IPv6_Route_t	118
6.1.2.6	struct qapi_Net_IPv6_Route_List_t	119
6.1.2.7	struct qapi_Net_Ifnameindex_t	119
6.1.2.8	struct qapi_Ping_Info_Resp_s	119
6.1.3	Enumeration Type Documentation	119
6.1.3.1	qapi_Net_Route_Command_t	119
6.1.3.2	qapi_Net_IPv4cfg_Command_t	120
6.2	Get the Names of All Network Interfaces	121
6.2.1	Function Documentation	121
6.2.1.1	qapi_Net_Get_All_Ifnames	121
6.3	Parse an Address String into an IPv4/IPv6 Address	122
6.3.1	Function Documentation	122
6.3.1.1	inet_pton	122
6.4	Format an IPv4/IPv6 Address into a NULL-terminated String	123
6.4.1	Function Documentation	123
6.4.1.1	inet_ntop	123
6.5	Get the Physical Address and Length of an Interface	124

6.5.1	Function Documentation	124
6.5.1.1	qapi_Net_Interface_Get_Physical_Address	124
6.6	Check Whether an Interface Exists	125
6.6.1	Function Documentation	125
6.6.1.1	qapi_Net_Interface_Exist	125
6.7	IPv4 Network Configuration	126
6.7.1	Function Documentation	126
6.7.1.1	qapi_Net_IPv4_Config	126
6.8	Send an IPv4 Ping	127
6.8.1	Function Documentation	127
6.8.1.1	qapi_Net_Ping	127
6.9	Send an IPv4 Ping with a Response	128
6.9.1	Function Documentation	128
6.9.1.1	qapi_Net_Ping_2	128
6.10	IPv4 Route Commands	129
6.10.1	Function Documentation	129
6.10.1.1	qapi_Net_IPv4_Route	129
6.11	Send an IPv6 Ping	130
6.11.1	Function Documentation	130
6.11.1.1	qapi_Net_Ping6	130
6.12	Send an IPv6 Ping with a Resonse	131
6.12.1	Function Documentation	131
6.12.1.1	qapi_Net_Ping6_2	131
6.13	Get the IPv6 Address of an Interface	132
6.13.1	Function Documentation	132
6.13.1.1	qapi_Net_IPv6_Get_Address	132
6.14	IPv6 Route Commands	133
6.14.1	Function Documentation	133
6.14.1.1	qapi_Net_IPv6_Route	133
6.15	Get the Interface Scope ID	134
6.15.1	Function Documentation	134
6.15.1.1	qapi_Net_IPv6_Get_Scope_ID	134
<b>7</b>	<b>Domain Name System Client Service APIs</b>	<b>135</b>
7.1	DNS Client Service Macros, Data Types, and Enumerations	136
7.1.1	Define Documentation	136
7.1.1.1	MAX_DNS_SVR_NUM	136
7.1.1.2	QAPI_DNS_PORT	136
7.1.1.3	QAPI_NET_DNS_ANY_SERVER_ID	136
7.1.1.4	gethostbyname	136
7.1.2	Data Structure Documentation	136
7.1.2.1	struct qapi_Net_DNS_Server_List_t	136
7.1.2.2	struct qapi_hostent_s	136
7.1.3	Enumeration Type Documentation	137
7.1.3.1	qapi_Net_DNS_Command_t	137
7.2	Check Whether the DNS Client has Started	138
7.2.1	Function Documentation	138
7.2.1.1	qapi_Net_DNSc_Is_Started	138
7.3	Start, Stop, or Disable the DNS Client	139
7.3.1	Function Documentation	139

7.3.1.1	qapi_Net_DNSc_Command	139
7.4	Convert an IP Address Text String into an IP Address	140
7.4.1	Function Documentation	140
7.4.1.1	qapi_Net_DNSc_Reshost	140
7.5	Convert an IP Address Text String for an Interface	141
7.5.1	Function Documentation	141
7.5.1.1	qapi_Net_DNSc_Reshost_on_iface	141
7.6	Get a List of DNS Servers	142
7.6.1	Function Documentation	142
7.6.1.1	qapi_Net_DNSc_Get_Server_List	142
7.7	Get Index for Added DNS Server	143
7.7.1	Function Documentation	143
7.7.1.1	qapi_Net_DNSc_Get_Server_Index	143
7.8	Add a DNS Server	144
7.8.1	Function Documentation	144
7.8.1.1	qapi_Net_DNSc_Add_Server	144
7.9	Add a DNS Server to an Interface	145
7.9.1	Function Documentation	145
7.9.1.1	qapi_Net_DNSc_Add_Server_on_iface	145
7.10	Remove a DNS Server	146
7.10.1	Function Documentation	146
7.10.1.1	qapi_Net_DNSc_Del_Server	146
7.11	Removes a DNS Server from an Interface	147
7.11.1	Function Documentation	147
7.11.1.1	qapi_Net_DNSc_Del_Server_on_iface	147
7.12	Get IPv4 Host Information by Name	148
7.12.1	Function Documentation	148
7.12.1.1	qapi_Net_DNSc_Get_Host_By_Name	148
7.13	Get IPv4/IPv6 Host Information by Name	149
7.13.1	Function Documentation	149
7.13.1.1	qapi_Net_DNSc_Get_Host_By_Name2	149
<b>8</b>	<b>MQTT API</b>	<b>150</b>
8.1	MQTT Data Types	151
8.1.1	Define Documentation	151
8.1.1.1	QAPI_NET_MQTT_MAX_CLIENT_ID_LEN	151
8.1.1.2	QAPI_NET_MQTT_MAX_TOPIC_LEN	151
8.1.2	Data Structure Documentation	151
8.1.2.1	struct qapi_Net_MQTT_config_s	151
8.1.3	Enumeration Type Documentation	152
8.1.3.1	QAPI_NET_MQTT_SUBSCRIBE_CBK_MSG	152
8.1.3.2	QAPI_NET_MQTT_CONNECT_CBK_MSG	152
8.1.3.3	QAPI_NET_MQTT_CONN_STATE	152
8.2	MQTT APIs	153
8.2.1	Function Documentation	153
8.2.1.1	qapi_Net_MQTT_New	153
8.2.1.2	qapi_Net_MQTT_Destroy	153
8.2.1.3	qapi_Net_MQTT_Connect	153
8.2.1.4	qapi_Net_MQTT_Disconnect	153
8.2.1.5	qapi_Net_MQTT_Publish	154

8.2.1.6	qapi_Net_MQTT_Subscribe	154
8.2.1.7	qapi_Net_MQTT_Unsubscribe	154
8.2.1.8	qapi_Net_MQTT_Set_Connect_Callback	155
8.2.1.9	qapi_Net_MQTT_Set_Subscribe_Callback	155
8.2.1.10	qapi_Net_MQTT_Set_Message_Callback	155
<b>9</b>	<b>HTTP(S) APIs</b>	<b>157</b>
9.1	HTTP(S) API	158
9.1.1	Data Structure Documentation	158
9.1.1.1	struct qapi_Net_HTTPc_Response_t	158
9.1.2	Typedef Documentation	158
9.1.2.1	qapi_HTTPc_CB_t	158
9.1.3	Enumeration Type Documentation	158
9.1.3.1	qapi_Net_HTTPc_Method_e	158
9.1.3.2	qapi_Net_HTTPc_CB_State_e	158
9.1.4	Function Documentation	159
9.1.4.1	qapi_Net_HTTPc_Start	159
9.1.4.2	qapi_Net_HTTPc_Stop	159
9.1.4.3	qapi_Net_HTTPc_New_sess	159
9.1.4.4	qapi_Net_HTTPc_Free_sess	160
9.1.4.5	qapi_Net_HTTPc_Connect	160
9.1.4.6	qapi_Net_HTTPc_Disconnect	161
9.1.4.7	qapi_Net_HTTPc_Request	161
9.1.4.8	qapi_Net_HTTPc_Set_Body	161
9.1.4.9	qapi_Net_HTTPc_Set_Param	162
9.1.4.10	qapi_Net_HTTPc_Add_Header_Field	162
9.1.4.11	qapi_Net_HTTPc_Clear_Header	162
9.1.4.12	qapi_Net_HTTPc_Configure_SSL	163
<b>10</b>	<b>QAPI Status and Error Codes</b>	<b>164</b>
10.1	QAPI Status Codes	165
10.1.1	Define Documentation	168
10.1.1.1	QAPI_ERR_SSL_CERT	168
10.1.1.2	QAPI_ERR_SSL_CONN	169
10.1.1.3	QAPI_ERR_SSL_HS_NOT_DONE	169
10.1.1.4	QAPI_ERR_SSL_ALERT_RECV	169
10.1.1.5	QAPI_ERR_SSL_ALERT_FATAL	169
10.1.1.6	QAPI_SSL_HS_IN_PROGRESS	169
10.1.1.7	QAPI_SSL_OK_HS	169
10.1.1.8	QAPI_ERR_SSL_CERT_CN	169
10.1.1.9	QAPI_ERR_SSL_CERT_TIME	169
10.1.1.10	QAPI_ERR_SSL_CERT_NONE	169
10.1.1.11	QAPI_ERR_SSL_NETBUF	169
10.1.1.12	QAPI_ERR_SSL SOCK	170
10.1.1.13	QAPI_NET_ERR_INVALID_IPADDR	170
10.1.1.14	QAPI_NET_ERR_CANNOT_GET_SCOPEID	170
10.1.1.15	QAPI_NET_ERR_SOCKET_CMD_TIME_OUT	170
10.1.1.16	QAPI_NET_ERR_MAX_SERVER_REACHED	170
10.1.1.17	QAPI_NET_MQTT_ERR_NUM_START	170
10.1.1.18	QAPI_NET_MQTT_ERR_ALLOC_FAILURE	170

10.1.1.19	QAPI_NET_MQTT_ERR_BAD_PARAM	170
10.1.1.20	QAPI_NET_MQTT_ERR_BAD_STATE	170
10.1.1.21	QAPI_NET_MQTT_ERR_CONN_CLOSED	171
10.1.1.22	QAPI_NET_MQTT_ERR_MSG_DESERIALIZATION_FAILURE	171
10.1.1.23	QAPI_NET_MQTT_ERR_MSG_SERIALIZATION_FAILURE	171
10.1.1.24	QAPI_NET_MQTT_ERR_NEGATIVE_CONNACK	171
10.1.1.25	QAPI_NET_MQTT_ERR_NO_DATA	171
10.1.1.26	QAPI_NET_MQTT_ERR_NONZERO_REFCOUNT	171
10.1.1.27	QAPI_NET_MQTT_ERR_PINGREQ_MSG_CREATION_FAILED	171
10.1.1.28	QAPI_NET_MQTT_ERR_PUBACK_MSG_CREATION_FAILED	171
10.1.1.29	QAPI_NET_MQTT_ERR_PUBCOMP_MSG_CREATION_FAILED	172
10.1.1.30	QAPI_NET_MQTT_ERR_PUBLISH_MSG_CREATION_FAILED	172
10.1.1.31	QAPI_NET_MQTT_ERR_PUBREC_MSG_CREATION_FAILED	172
10.1.1.32	QAPI_NET_MQTT_ERR_PUBREL_MSG_CREATION_FAILED	172
10.1.1.33	QAPI_NET_MQTT_ERR_RX_INCOMPLETE	172
10.1.1.34	QAPI_NET_MQTT_ERR_SOCKET_FATAL_ERROR	172
10.1.1.35	QAPI_NET_MQTT_ERR_TCP_BIND_FAILED	172
10.1.1.36	QAPI_NET_MQTT_ERR_TCP_CONNECT_FAILED	172
10.1.1.37	QAPI_NET_MQTT_ERR_SSL_CONN_FAILURE	173
10.1.1.38	QAPI_NET_MQTT_ERR_SUBSCRIBE_MSG_CREATION_FAILED	173
10.1.1.39	QAPI_NET_MQTT_ERR_SUBSCRIBE_UNKNOWN_TOPIC	173
10.1.1.40	QAPI_NET_MQTT_ERR_UNSUBSCRIBE_MSG_CREATION_FAILED	173
10.1.1.41	QAPI_NET_MQTT_ERR_UNEXPECTED_MSG	173
10.1.1.42	QAPI_NET_MQTT_ERR_PARTIAL_SUBSCRIPTION_FAILURE	173
10.1.1.43	QAPI_NET_MQTT_ERR_RESTORE_FAILURE	173
10.1.1.44	QAPI_NET_MQTT_ERR_MAX_NUMS	173
10.1.1.45	QAPI_OK	174
10.1.1.46	QAPI_ERROR	174
10.1.1.47	QAPI_ERR_INVALID_PARAM	174
10.1.1.48	QAPI_ERR_NO_MEMORY	174
10.1.1.49	QAPI_ERR_NO_RESOURCE	174
10.1.1.50	QAPI_ERR_BUSY	174
10.1.1.51	QAPI_ERR_NO_ENTRY	174
10.1.1.52	QAPI_ERR_NOT_SUPPORTED	174
10.1.1.53	QAPI_ERR_TIMEOUT	174
10.1.1.54	QAPI_ERR_BOUNDS	174
10.1.1.55	QAPI_ERR_BAD_PAYLOAD	175
10.1.1.56	QAPI_ERR_EXISTS	175
<b>11</b>	<b>System Drivers APIs</b>	<b>176</b>
11.1	GPIO Interrupt Controller APIs	177
11.1.1	Typedef Documentation	178
11.1.1.1	qapi_GPIINT_Callback_Data_t	178
11.1.1.2	qapi_GPIINT_CB_t	178
11.1.1.3	qapi_Instance_Handle_t	178
11.1.2	Enumeration Type Documentation	178
11.1.2.1	qapi_GPIINT_Trigger_e	178
11.1.2.2	qapi_GPIINT_Priority_e	179
11.1.3	Function Documentation	179
11.1.3.1	qapi_GPIINT_Register_Interrupt	179



11.1.3.2	qapi_GPIPOINT_Deregister_Interrupt	180
11.1.3.3	qapi_GPIPOINT_Set_Trigger	180
11.1.3.4	qapi_GPIPOINT_Enable_Interrupt	180
11.1.3.5	qapi_GPIPOINT_Disable_Interrupt	181
11.1.3.6	qapi_GPIPOINT_Trigger_Interrupt	181
11.1.3.7	qapi_GPIPOINT_Is_Interrupt_Pending	182
11.2	PMM APIs	183
11.2.1	Data Structure Documentation	184
11.2.1.1	struct qapi_TLMM_Config_t	184
11.2.2	Typedef Documentation	184
11.2.2.1	qapi_GPIO_ID_t	184
11.2.3	Enumeration Type Documentation	184
11.2.3.1	qapi_GPIO_Direction_t	184
11.2.3.2	qapi_GPIO_Pull_t	184
11.2.3.3	qapi_GPIO_Drive_t	185
11.2.3.4	qapi_GPIO_Value_t	185
11.2.4	Function Documentation	185
11.2.4.1	qapi_TLMM_Get_Gpio_ID	185
11.2.4.2	qapi_TLMM_Release_Gpio_ID	186
11.2.4.3	qapi_TLMM_Config_Gpio	186
11.2.4.4	qapi_TLMM_Drive_Gpio	187
11.2.4.5	qapi_TLMM_Read_Gpio	187
<b>12</b>	<b>Diagnostic Services Module</b>	<b>188</b>
12.1	QAPI Diag Services APIs	189
12.1.1	Define Documentation	189
12.1.1.1	QAPI_DIAGPKT_DISPATCH_TABLE_REGISTER	189
12.1.1.2	QAPI_DIAGPKT_DISPATCH_TABLE_REGISTER_V2_DELAY	189
12.1.1.3	QAPI_MSG	190
12.1.1.4	QAPI_MSG_SPRINTF	190
12.1.2	Data Structure Documentation	190
12.1.2.1	struct qapi_diagpkt_subsys_hdr_type_t	190
12.1.2.2	struct qapi_diagpkt_subsys_hdr_type_v2_t	191
12.1.3	Function Documentation	191
12.1.3.1	qapi_user_space_tbl_reg_append_proc	191
12.1.3.2	qapi_diagpkt_get_next_delayed_rsp_id	192
12.1.3.3	qapi_diagpkt_commit	192
12.1.3.4	qapi_msg_send	192
12.1.3.5	qapi_msg_sprintf	193
12.1.3.6	qapi_log_submit	193
12.1.3.7	qapi_log_set_length	193
12.1.3.8	qapi_log_set_code	193
12.1.3.9	qapi_log_set_timestamp	194
12.1.3.10	qapi_log_status	194
12.1.3.11	qapi_event_report	194
12.1.3.12	qapi_event_report_payload	195
<b>13</b>	<b>Storage Module</b>	<b>196</b>
13.1	File System Data Types	197
13.1.1	Data Structure Documentation	197



13.1.1.1	struct qapi_FS_Stat_Type_s	197
13.1.1.2	struct qapi_FS_Statvfs_Type_s	197
13.1.1.3	struct qapi_FS_Iter_Entry_s	198
13.1.2	Enumeration Type Documentation	199
13.1.2.1	qapi_FS_Filename_Rule_e	199
13.1.2.2	qapi_FS_Filename_Encoding_e	199
13.2	File System APIs	200
13.2.1	Function Documentation	200
13.2.1.1	qapi_FS_Open_With_Mode	200
13.2.1.2	qapi_FS_Open	201
13.2.1.3	qapi_FS_Read	202
13.2.1.4	qapi_FS_Write	202
13.2.1.5	qapi_FS_Close	203
13.2.1.6	qapi_FS_Rename	203
13.2.1.7	qapi_FS_Truncate	203
13.2.1.8	qapi_FS_Seek	204
13.2.1.9	qapi_FS_Mk_Dir	205
13.2.1.10	qapi_FS_Rm_Dir	205
13.2.1.11	qapi_FS_Unlink	205
13.2.1.12	qapi_FS_Stat	206
13.2.1.13	qapi_FS_Stat_With_Handle	206
13.2.1.14	qapi_FS_Statvfs	207
13.2.1.15	qapi_FS_Iter_Open	207
13.2.1.16	qapi_FS_Iter_Next	207
13.2.1.17	qapi_FS_Iter_Close	208
13.2.1.18	qapi_FS_Last_Error	209
13.3	FTL Data Types and APIs	210
13.3.1	Data Structure Documentation	210
13.3.1.1	struct qapi_FTL_info	210
13.3.2	Typedef Documentation	210
13.3.2.1	qapi_FTL_info_t	210
13.3.2.2	qapi_FTL_client_t	210
13.3.3	Function Documentation	210
13.3.3.1	qapi_FTL_Open	210
13.3.3.2	qapi_FTL_Close	211
13.3.3.3	qapi_FTL_Get_info	211
13.3.3.4	qapi_FTL_Read_lpa	212
13.3.3.5	qapi_FTL_Write_lpa	212
13.3.3.6	qapi_FTL_Erase_block	213
<b>14</b>	<b>Wired Connectivity Module</b>	<b>214</b>
14.1	USB Data Types	215
14.1.1	Data Structure Documentation	215
14.1.1.1	union qapi_USB_ioctl_Param	215
14.1.2	Typedef Documentation	215
14.1.2.1	qapi_USB_App_Rx_Cb_t	215
14.1.2.2	qapi_USB_ioctl_Param_t	215
14.1.3	Enumeration Type Documentation	215
14.1.3.1	qapi_USB_ioctl_Cmd_t	215
14.2	USB APIs	216

14.2.1	Function Documentation	216
14.2.1.1	qapi_USB_Open	216
14.2.1.2	qapi_USB_Read	216
14.2.1.3	qapi_USB_Write	217
14.2.1.4	qapi_USB_ioctl	217
<b>15</b>	<b>Buses Module</b>	<b>218</b>
15.1	I2C Master APIs	219
15.1.1	Define Documentation	220
15.1.1.1	QAPI_I2C_FLAG_START	220
15.1.1.2	QAPI_I2C_FLAG_STOP	220
15.1.1.3	QAPI_I2C_FLAG_WRITE	221
15.1.1.4	QAPI_I2C_FLAG_READ	221
15.1.1.5	QAPI_I2C_TRANSFER_MASK	221
15.1.1.6	QAPI_VALID_FLAGS	221
15.1.2	Data Structure Documentation	221
15.1.2.1	struct qapi_I2CM_Config_t	221
15.1.2.2	struct qapi_I2CM_Descriptor_t	221
15.1.3	Typedef Documentation	222
15.1.3.1	qapi_I2CM_Transfer_CB_t	222
15.1.4	Enumeration Type Documentation	222
15.1.4.1	qapi_I2CM_Instance_t	222
15.1.5	Function Documentation	223
15.1.5.1	qapi_I2CM_Open	223
15.1.5.2	qapi_I2CM_Close	224
15.1.5.3	qapi_I2CM_Transfer	224
15.1.5.4	qapi_I2CM_Power_On	225
15.1.5.5	qapi_I2CM_Power_Off	225
15.2	SPI Master APIs	227
15.2.1	Data Structure Documentation	227
15.2.1.1	struct qapi_SPIM_Config_t	227
15.2.1.2	struct qapi_SPIM_Descriptor_t	228
15.2.2	Typedef Documentation	228
15.2.2.1	qapi_SPIM_Callback_Fn_t	228
15.2.3	Enumeration Type Documentation	229
15.2.3.1	qapi_SPIM_Instance_t	229
15.2.3.2	qapi_SPIM_Shift_Mode_t	229
15.2.3.3	qapi_SPIM_CS_Polarity_t	230
15.2.3.4	qapi_SPIM_Byte_Order_t	230
15.2.3.5	qapi_SPIM_CS_Mode_t	230
15.2.4	Function Documentation	231
15.2.4.1	qapi_SPIM_Open	231
15.2.4.2	qapi_SPIM_Power_On	231
15.2.4.3	qapi_SPIM_Power_Off	232
15.2.4.4	qapi_SPIM_Full_Duplex	232
15.2.4.5	qapi_SPIM_Close	233
15.3	UART APIs	234
15.3.1	Data Structure Documentation	234
15.3.1.1	union QAPI_UART_ioctl_Param	234
15.3.1.2	struct qapi_UART_Open_Config_t	234

15.3.2	Typedef Documentation	235
15.3.2.1	QAPI_UART_ioctl_Param	235
15.3.2.2	qapi_UART_Handle_t	235
15.3.2.3	qapi_UART_Callback_Fn_t	235
15.3.3	Enumeration Type Documentation	235
15.3.3.1	qapi_UART_Port_Id_e	235
15.3.3.2	qapi_UART_Bits_Per_Char_e	236
15.3.3.3	qapi_UART_Num_Stop_Bits_e	236
15.3.3.4	qapi_UART_Parity_Mode_e	236
15.3.3.5	qapi_UART_ioctl_Command_e	236
15.3.3.6	QAPI_Flow_Control_Type	237
15.3.4	Function Documentation	237
15.3.4.1	qapi_UART_Close	237
15.3.4.2	qapi_UART_Open	237
15.3.4.3	qapi_UART_Receive	238
15.3.4.4	qapi_UART_Transmit	238
15.3.4.5	qapi_UART_Power_On	239
15.3.4.6	qapi_UART_Power_Off	239
15.3.4.7	qapi_UART_ioctl	239
<b>16</b>	<b>Location Module</b>	<b>241</b>
16.1	Location APIs	242
16.1.1	Data Structure Documentation	242
16.1.1.1	struct qapi_Location_Options_t	242
16.1.1.2	struct qapi_Geofence_Options_t	242
16.1.1.3	struct qapi_Geofence_Data_t	242
16.1.1.4	struct qapi_Location_t	242
16.1.1.5	struct qapi_Location_Callbacks_t	243
16.1.2	Typedef Documentation	243
16.1.2.1	qapi_Capabilities_Callback	243
16.1.2.2	qapi_Response_Callback	244
16.1.2.3	qapi_Geofence_Response_Callback	244
16.1.2.4	qapi_Tracking_Callback	244
16.1.2.5	qapi_Batching_Callback	245
16.1.2.6	qapi_Geofence_Breach_Callback	245
16.1.3	Enumeration Type Documentation	245
16.1.3.1	qapi_Location_Error_t	245
16.1.3.2	qapi_Location_Flags_t	246
16.1.3.3	qapi_Geofence_Breach_t	246
16.1.3.4	qapi_Geofence_Breach_Mask_t	246
16.1.3.5	qapi_Location_Capabilities_Mask_t	246
16.1.4	Function Documentation	247
16.1.4.1	qapi_Loc_Init	247
16.1.4.2	qapi_Loc_Deinit	247
16.1.4.3	qapi_Loc_Start_Tracking	247
16.1.4.4	qapi_Loc_Stop_Tracking	248
16.1.4.5	qapi_Loc_Update_Tracking_Options	248
16.1.4.6	qapi_Loc_Start_Batching	248
16.1.4.7	qapi_Loc_Stop_Batching	249
16.1.4.8	qapi_Loc_Update_Batching_Options	249

16.1.4.9	qapi_Loc_Get_Batched_Locations	250
16.1.4.10	qapi_Loc_Add_Geofences	250
16.1.4.11	qapi_Loc_Remove_Geofences	251
16.1.4.12	qapi_Loc_Modify_Geofences	251
16.1.4.13	qapi_Loc_Pause_Geofences	251
16.1.4.14	qapi_Loc_Resume_Geofences	252
<b>17</b>	<b>Timer Module</b>	<b>253</b>
17.1	Timer APIs	254
17.1.1	Data Structure Documentation	254
17.1.1.1	struct qapi_TIMER_define_attr_t	254
17.1.1.2	struct qapi_TIMER_set_attr_t	255
17.1.1.3	struct qapi_TIMER_get_info_attr_t	255
17.1.2	Typedef Documentation	255
17.1.2.1	qapi_TIMER_handle_t	255
17.1.2.2	qapi_TIMER_cb_t	256
17.1.3	Enumeration Type Documentation	256
17.1.3.1	qapi_TIMER_notify_t	256
17.1.3.2	qapi_TIMER_unit_type	256
17.1.3.3	qapi_TIMER_info_type	256
17.1.4	Function Documentation	257
17.1.4.1	qapi_Timer_Def	257
17.1.4.2	qapi_Timer_Set	257
17.1.4.3	qapi_Timer_Get_Timer_Info	257
17.1.4.4	qapi_Timer_Sleep	258
17.1.4.5	qapi_Timer_Undef	258
17.1.4.6	qapi_Timer_Stop	258
17.1.4.7	qapi_Timer_set_absolute	259
17.2	PMIC RTC APIs	260
17.2.1	Data Structure Documentation	260
17.2.1.1	struct qapi_PM_Rtc_Julian_Type_s	260
17.2.2	Typedef Documentation	260
17.2.2.1	qapi_PM_Rtc_Julian_Type_t	260
17.2.3	Enumeration Type Documentation	260
17.2.3.1	qapi_PM_Rtc_Cmd_Type_t	260
17.2.3.2	qapi_PM_Rtc_Display_Type_t	260
17.2.3.3	qapi_PM_Rtc_Alarm_Type_t	261
17.2.4	Function Documentation	261
17.2.4.1	qapi_PM_Rtc_Init	261
17.2.4.2	qapi_PM_Set_Rtc_Display_Mode	261
17.2.4.3	qapi_PM_Rtc_Read_Cmd	262
17.2.4.4	qapi_PM_Rtc_Alarm_RW_Cmd	262
<b>18</b>	<b>Hardware Engine APIs</b>	<b>264</b>
18.1	ADC Data Types	265
18.1.1	Define Documentation	265
18.1.1.1	ADC_INPUT_BATT_ID	265
18.1.1.2	ADC_INPUT_PA_THERM	265
18.1.1.3	ADC_INPUT_PA_THERM1	265
18.1.1.4	ADC_INPUT_PMIC_THERM	265

18.1.1.5	ADC_INPUT_VBATT	265
18.1.1.6	ADC_INPUT_VPH_PWR	265
18.1.1.7	ADC_INPUT_XO_THERM	265
18.1.1.8	ADC_INPUT_XO_THERM_GPS	265
18.1.2	Data Structure Documentation	265
18.1.2.1	struct qapi_ADC_Read_Result_t	265
18.1.2.2	struct qapi_Adc_Input_Properties_Type_t	266
18.1.2.3	struct qapi_AdcTM_Input_Properties_Type_t	266
18.1.2.4	struct qapi_ADC_Range_t	266
18.1.2.5	struct qapi_ADC_Threshold_Result_t	266
18.1.2.6	struct qapi_ADC_Device_Properties_t	267
18.1.2.7	struct qapi_AdcTM_Callback_Payload_Type_t	267
18.1.2.8	struct qapi_AdcTM_Range_Type_t	267
18.1.2.9	struct qapi_AdcTM_Request_Params_Type_t	267
18.1.3	Typedef Documentation	268
18.1.3.1	qapi_ADC_Threshold_CB_t	268
18.1.3.2	qapi_AdcTM_Threshold_Cb_Type	268
18.1.4	Enumeration Type Documentation	268
18.1.4.1	qapi_ADC_Amp_Threshold_t	268
18.2	ADC APIs	269
18.2.1	Function Documentation	270
18.2.1.1	qapi_ADC_Open	270
18.2.1.2	qapi_ADC_Get_Input_Properties	270
18.2.1.3	qapi_ADC_Read_Channel	271
18.2.1.4	qapi_ADC_TM_Get_Input_Properties	271
18.2.1.5	qapi_ADC_Get_Range	272
18.2.1.6	qapi_ADC_Set_Amp_Threshold	272
18.2.1.7	qapi_ADC_TM_Enable_Thresholds	273
18.2.1.8	qapi_ADC_TM_Set_Tolerance	273
18.2.1.9	qapi_ADC_Close	274
18.3	TSENS Data Types	275
18.3.1	Data Structure Documentation	275
18.3.1.1	struct qapi_TSENS_CallbackPayloadType_t	275
18.3.1.2	struct qapi_TSENS_Result_t	275
18.3.2	Typedef Documentation	276
18.3.2.1	QAPI_Tsens_Threshold_Cb_Type	276
18.3.2.2	qapi_TSENS_Handle_t	276
18.3.3	Enumeration Type Documentation	276
18.3.3.1	qapi_TSENS_ThresholdType_t	276
18.4	TSENS APIs	277
18.4.1	Function Documentation	278
18.4.1.1	qapi_TSENS_Open	278
18.4.1.2	qapi_TSENS_Get_Num_Sensors	278
18.4.1.3	qapi_TSENS_Get_Temp	278
18.4.1.4	qapi_TSENS_Get_Calibration_Status	279
18.4.1.5	qapi_TSENS_Set_Thresholds	279
18.4.1.6	qapi_TSENS_Set_Enable_Thresholds	280
18.4.1.7	qapi_TSENS_Close	280

## 19 System Power Save Management

282

19.1 PSM Data Types	283
19.1.1 Define Documentation	283
19.1.1.1 QAPI_ERR_PSM_FAIL	283
19.1.1.2 QAPI_ERR_PSM_GENERIC_FAILURE	283
19.1.1.3 QAPI_ERR_PSM_APP_NOT_REGISTERED	283
19.1.1.4 QAPI_ERR_PSM_WRONG_ARGUMENTS	283
19.1.1.5 QAPI_ERR_PSM_IPC_FAILURE	283
19.1.2 Data Structure Documentation	283
19.1.2.1 struct psm_time_info_type	283
19.1.2.2 struct psm_info_type	284
19.1.2.3 struct psm_status_msg_type	284
19.1.3 Typedef Documentation	284
19.1.3.1 psm_client_cb_type	284
19.1.3.2 psm_util_timer_expiry_cb_type	284
19.1.4 Enumeration Type Documentation	284
19.1.4.1 psm_status_type_e	284
19.1.4.2 psm_reject_reason_type_e	285
19.1.4.3 psm_error_type_e	285
19.1.4.4 psm_time_format_type_e	285
19.1.4.5 psm_wakeup_type_e	285
<b>20 Device Information Module</b>	<b>286</b>
20.1 Device Information	287
20.1.1 Define Documentation	287
20.1.1.1 QAPI_DEVICE_INFO_BUF_SIZE	287
20.1.2 Data Structure Documentation	287
20.1.2.1 struct qapi_Device_Info_t	287
20.1.2.2 union qapi_Device_Info_t.u	287
20.1.2.3 struct qapi_Device_Info_t.u.valuebuf	287
20.1.3 Enumeration Type Documentation	288
20.1.3.1 qapi_Device_Info_ID_t	288
20.1.3.2 qapi_Device_Info_Type_t	288
20.1.4 Function Documentation	289
20.1.4.1 qapi_Device_Info_Init	289
20.1.4.2 qapi_Device_Info_Get	289
20.1.4.3 qapi_Device_Info_Set_Callback	289
20.1.4.4 qapi_Device_Info_Release	290
20.1.4.5 qapi_Device_Info_Reset	290
20.2 Qapi_net_datatypes	291
<b>21 Use Cases</b>	<b>292</b>
21.1 DSS API Use Case	292
21.2 Socket API Use Cases	292
21.2.1 Server Socket	292
21.2.2 Client Socket	293
21.3 TLS/DTLS API Use Cases	294
21.3.1 TLS/DTLS Context Object Creation	294
21.3.2 TLS/DTLS Certificate or CA List, or PSK Table Store and Load to SSL Context	295
21.3.3 TLS/DTLS Connection Object Creation	295
21.3.4 TLS/DTLS Configuration of a Connection Object	295

21.3.5 Secure Socket Data Transfer over a TLS/DTLS Connection . . . . .	296
21.3.6 Close an SSL Connection TLS/DTLS Connection and Socket . . . . .	297
21.3.7 TLS/DTLS Close Context Object . . . . .	298
<b>A References</b> . . . . .	<b>299</b>
A.1 Related Documents . . . . .	299
A.2 Acronyms and Terms . . . . .	299

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## List of Figures

2-1 Data call architecture in ThreadX OS . . . . .	26
--	----

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com



# 1 Introduction

---

## 1.1 Purpose

This document is the reference specification for the Qualcomm Application Programming Interface (QAPI) for the MDM9206 ThreadX OS.

The QAPIs are designed to facilitate the development of mobile station-based networking applications.

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

## 1.2 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font, e.g., `#include`.

## 1.3 Technical Assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at <https://support.cdmatech.com>.

If you do not have access to the CDMATech Support website, register for access or send email to [support.cdmatech@qti.qualcomm.com](mailto:support.cdmatech@qti.qualcomm.com).

## 2 Data Call Functional Overview

Data call establishment on the ThreadX OS is achieved using QAPIs. QAPIs are used to establish the control plane to set routes and DNS information and expose BSD-style socket APIs for data transfer. The following section describes the data call architecture, and Chapter 3 contains details on each of the QAPIs.

### 2.1 Data call architecture in the ThreadX OS

In the block diagram (Figure 2-1), the blue line maps the control path while the red line marks the data path. A typical application, such as Data Connection Manager, triggers a call request using QAPIs, resulting in the return of an IP address assigned by the WWAN network. Upon receiving the indication, the DSS layer configures a network interface with the IP address. A maximum of four interfaces are supported. This completes a control path. Using the QAPIs, the Connection Manager can establish a socket connection for sending and receiving data.

Note that only address assignments are handled automatically. Routes and DNS address configuration is left to the application to configure.

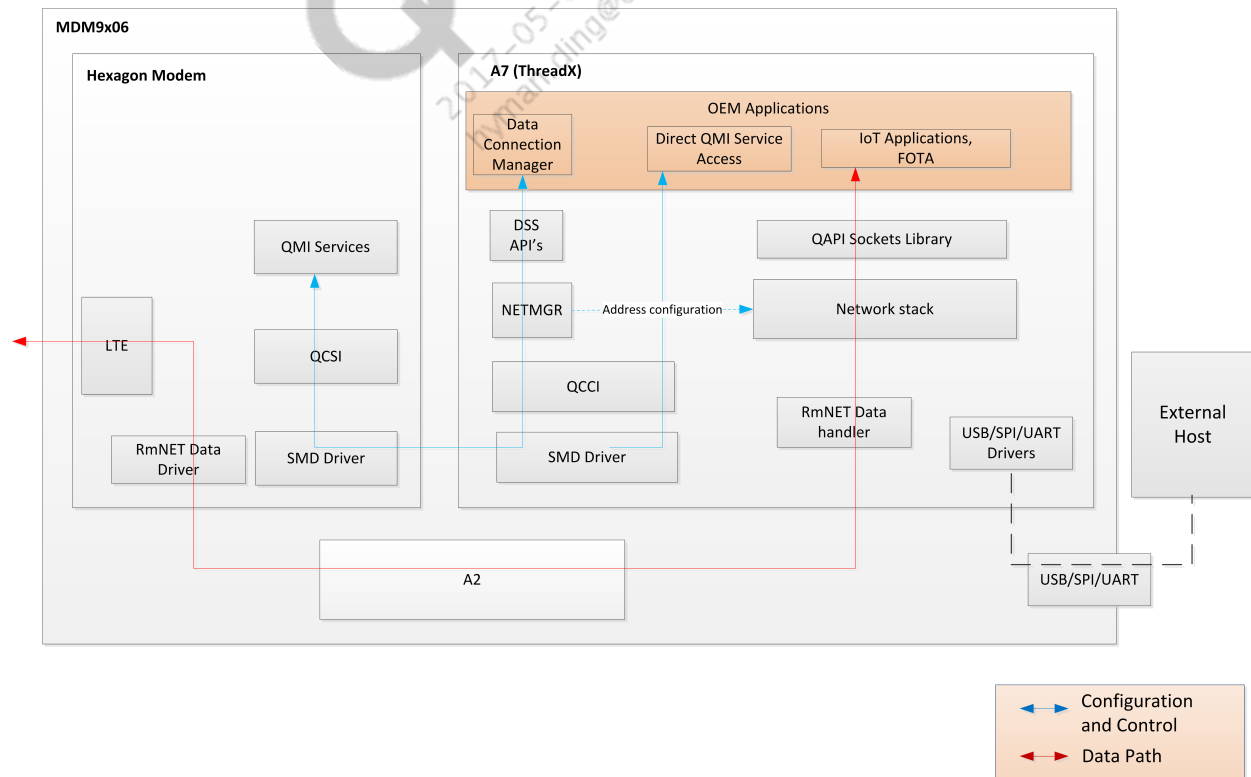


Figure 2-1 Data call architecture in ThreadX OS

# 3 DSS Net Control APIs

---

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

- [DSS Netctrl Macros, Data Structures, and Enumerations](#)
- [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](#)

## 3.1 DSS Netctrl Macros, Data Structures, and Enumerations

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

### 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_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

### 3.1.1 Define Documentation

#### 3.1.1.1 #define QAPI\_DSS\_RADIO\_TECH\_UNKNOWN 0x00000000

Technology is unknown.

#### 3.1.1.2 #define QAPI\_DSS\_RADIO\_TECH\_MIN 0x00000001

Start.

#### 3.1.1.3 #define QAPI\_DSS\_RADIO\_TECH\_UMTS QAPI\_DSS\_RADIO\_TECH\_MIN

UMTS.

#### 3.1.1.4 #define QAPI\_DSS\_RADIO\_TECH\_CDMA 0x00000002

CDMA.

#### 3.1.1.5 #define QAPI\_DSS\_RADIO\_TECH\_1X 0x00000004

1X.

#### 3.1.1.6 #define QAPI\_DSS\_RADIO\_TECH\_DO 0x00000008

DO.

#### 3.1.1.7 #define QAPI\_DSS\_RADIO\_TECH\_LTE 0x00000010

LTE.

#### 3.1.1.8 #define QAPI\_DSS\_RADIO\_TECH\_TDSCDMA 0x00000020

TDSCDMA.

#### 3.1.1.9 #define QAPI\_DSS\_CALL\_INFO\_USERNAME\_MAX\_LEN 127

Maximum length of the username.

#### 3.1.1.10 #define QAPI\_DSS\_CALL\_INFO\_PASSWORD\_MAX\_LEN 127

Maximum length of the password.

#### 3.1.1.11 #define QAPI\_DSS\_CALL\_INFO\_APN\_MAX\_LEN 150

Maximum length of the APN.

**3.1.1.12 #define QAPI\_DSS\_CALL\_INFO\_DEVICE\_NAME\_MAX\_LEN 12**

Maximum length of the device name.

**3.1.1.13 #define QAPI\_DSS\_SUCCESS 0**

Indicates that the operation was successful.

**3.1.1.14 #define QAPI\_DSS\_ERROR -1**

Indicates that the operation was not successful.

**3.1.1.15 #define QAPI\_DSS\_IP\_VERSION\_4 4**

IP version v4.

**3.1.1.16 #define QAPI\_DSS\_IP\_VERSION\_6 6**

IP version v6.

**3.1.1.17 #define QAPI\_DSS\_IP\_VERSION\_4\_6 10**

IP version v4v6.

**3.1.1.18 #define QAPI\_DSS\_MODE\_GENERAL 0**

General mode.

**3.1.2 Data Structure Documentation****3.1.2.1 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.

**3.1.2.2 struct qapi\_DSS\_Call\_Param\_Value\_s****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.

### 3.1.2.3 struct qapi\_DSS\_Addr\_s

#### Data fields

Type	Parameter	Description
char	valid_addr	Indicates whether a valid address is available.
union qapi_dss-_ip_address_u	addr	

### 3.1.2.4 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.

### 3.1.2.5 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.

### 3.1.2.6 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.

Type	Parameter	Description
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.

### 3.1.3 Typedef Documentation

#### 3.1.3.1 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.

### 3.1.4 Enumeration Type Documentation

#### 3.1.4.1 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.

#### 3.1.4.2 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.

### 3.1.4.3 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.

### 3.1.4.4 enum qapi\_DSS\_Net\_Evt\_e

Event names, which 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.

### 3.1.4.5 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.

### 3.1.4.6 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\_CDMA\_1X\_E** 1X technology.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_EVDO\_REV0\_E** CDMA Rev 0.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_EVDO\_REVA\_E** CDMA Rev A.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_EVDO\_REVB\_E** CDMA Rev B.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_EHRPD\_E** EHRPD.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_FMC\_E** Fixed mobile convergence.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_HRPD\_E** HRPD.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_3GPP2\_WLAN\_E** IWLAN.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_WCDMA\_E** WCDMA.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_GPRS\_E** GPRS.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_HSDPA\_E** HSDPA.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_HSUPA\_E** HSUPA.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_EDGE\_E** EDGE.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_LTE\_E** LTE.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_HSDPA\_PLUS\_E** HSDPA+.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_DC\_HSDPA\_PLUS\_E** DC HSDPA+.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_HSPA\_E** HSPA.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_64\_QAM\_E** 64 QAM.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_TDSCDMA\_E** TD-SCDMA.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_GSM\_E** GSM.  
**QAPI\_DSS\_DATA\_BEARER\_TECH\_3GPP\_WLAN\_E** IWLAN.

**3.1.4.7 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.

## 3.2 Initialize the DSS Netctrl Library

### 3.2.1 Function Documentation

#### 3.2.1.1 `qapi_Status_t qapi_DSS_Init ( int mode )`

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

**Note:** Only QAPI\_DSS\_MODE\_GENERAL is to be used by applications.

##### Parameters

<code>in</code>	<code>mode</code>	Mode of operation in which to initialize the library.
-----------------	-------------------	---

##### Returns

- QAPI\_OK - Initialization successful.
- QAPI\_ERROR - Initialization failed.

## 3.3 Release the DSS Netctrl Library

### 3.3.1 Function Documentation

#### 3.3.1.1 `qapi_Status_t qapi_DSS_Release ( int mode )`

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

**Note:** Only QAPI\_DSS\_MODE\_GENERAL is to be used by applications.

##### Parameters

<code>in</code>	<code>mode</code>	Mode of operation in which to de-initialize the library.
-----------------	-------------------	--

##### Returns

- QAPI\_OK – Cleanup was successful.
- QAPI\_ERROR – Cleanup failed.

## 3.4 Get the Data Service Handle

### 3.4.1 Function Documentation

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

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

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

#### Parameters

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

#### Returns

- QAPI\_OK – Cleanup was successful.
- QAPI\_ERROR – Cleanup failed.

#### Dependencies

[`qapi\_DSS\_Init\(\)`](#) must have been called first.

## 3.5 Release the Data Service Handle

### 3.5.1 Function Documentation

#### 3.5.1.1 `qapi_Status_t qapi_DSS_Rel_Data_Srvc_Hndl ( qapi_DSS_Hndl_t hndl )`

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

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

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srvc_Hndl()</a> .
----	-------------	--

##### Returns

- QAPI\_OK – Cleanup was successful.
- QAPI\_ERROR – Cleanup failed.

##### Dependencies

[qapi\\_DSS\\_Init\(\)](#) must have been called first.

A valid handle must be obtained by [qapi\\_DSS\\_Get\\_Data\\_Srvc\\_Hndl\(\)](#).

## 3.6 Set the Data Call Parameter

### 3.6.1 Function Documentation

**3.6.1.1** `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 )`

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.

#### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srvc_Hndl()</a> .
in	<i>identifier</i>	Identifies the parameter information.
in	<i>info</i>	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\(\)](#).

## 3.7 Start a Data Call

### 3.7.1 Function Documentation

#### 3.7.1.1 `qapi_Status_t qapi_DSS_Start_Data_Call ( qapi_DSS_Hndl_t hndl )`

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.

##### Parameters

in	<i>hndl</i>	Handle received from <code>qapi_DSS_Get_Data_Srvc_Hndl()</code> .
----	-------------	---

##### 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()`.



## 3.8 Stop a Data Call

### 3.8.1 Function Documentation

#### 3.8.1.1 `qapi_Status_t qapi_DSS_Stop_Data_Call ( qapi_DSS_Hndl_t hndl )`

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_Srv_Hndl()` indicating the data call tear-down status.

##### Parameters

in	<i>hndl</i>	Handle received from <code>qapi_DSS_Get_Data_Srv_Hndl()</code> .
----	-------------	--

##### 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_Srv_Hndl()`.

The data call must have been brought up using `qapi_DSS_Start_Data_Call()`.

## 3.9 Get Packet Data Transfer Statistics

### 3.9.1 Function Documentation

#### 3.9.1.1 `qapi_Status_t qapi_DSS_Get_Pkt_Stats ( qapi_DSS_Hndl_t hndl, qapi_DSS_Data_Pkt_Stats_t * dss_data_stats )`

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

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srv_Hndl()</a> .
in	<i>dss_data_stats</i>	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\\_Srv\\_Hndl\(\)](#).

## 3.10 Reset Packet Data Transfer Statistics

### 3.10.1 Function Documentation

#### 3.10.1.1 `qapi_Status_t qapi_DSS_Reset_Pkt_Stats ( qapi_DSS_Hndl_t hndl )`

Resets the packet data transfer statistics.

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srvc_Hndl()</a> .
----	-------------	--

##### 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\(\)](#).

## 3.11 Get the Data Call End Reason

### 3.11.1 Function Documentation

#### 3.11.1.1 `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 )`

Queries for the reason a data call was ended.

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srv_Hndl()</a> .
out	<i>ce_reason</i>	Buffer to hold data call ending reason information.
in	<i>ip_family</i>	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\(\)](#).

## 3.12 Get the Data Call Technology

### 3.12.1 Function Documentation

#### 3.12.1.1 `qapi_Status_t qapi_DSS_Get_Call_Tech ( qapi_DSS_Hndl_t hndl, qapi_DSS_Call_Tech_Type_t * 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.

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srv_Hndl()</a> .
out	<i>call_tech</i>	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\(\)](#).

## 3.13 Get the Data Bearer Technology

### 3.13.1 Function Documentation

#### 3.13.1.1 `qapi_Status_t qapi_DSS_Get_Current_Data_Bearer_Tech ( qapi_DSS_Hndl_t hndl, qapi_DSS_Data_Bearer_Tech_t * 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.

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srvc_Hndl()</a> .
in	<i>bearer_tech</i>	Pointer 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\\_Srvc\\_Hndl\(\)](#).

## 3.14 Get the Device Name

### 3.14.1 Function Documentation

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

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

**Note:** *len* must be at least QAPI\_DSS\_CALL\_INFO\_DEVICE\_NAME\_MAX\_LEN + 1 long.

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srv_Hndl()</a> .
out	<i>buf</i>	Buffer to hold the data interface name string.
in	<i>len</i>	Length of the buffer allocated by 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\\_Srv\\_Hndl\(\)](#).

## 3.15 Get the QMI Port Name

### 3.15.1 Function Documentation

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

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

**Note:** *len* must be at least DSI\_CALL\_INFO\_DEVICE\_NAME\_MAX\_LEN + 1 long.

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srv_Hndl()</a> .
out	<i>buf</i>	Buffer to hold the QMI port name string.
in	<i>len</i>	Length of the buffer allocated by 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\\_Srv\\_Hndl\(\)](#).



## 3.16 Get the IP Address Count

### 3.16.1 Function Documentation

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

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

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srvc_Hndl()</a> .
in	<i>ip_addr_cnt</i>	Pointer to retrieve 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\\_Srvc\\_Hndl\(\)](#).

## 3.17 Get the IP Address Information

### 3.17.1 Function Documentation

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

Queries the IP address information structure (network order).

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srvc_Hndl()</a> .
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\\_Srvc\\_Hndl\(\)](#).

The length parameter can be obtained by calling [qapi\\_DSS\\_Get\\_IP\\_Addr\\_Count\(\)](#).

It is assumed that the client has allocated memory for enough structures specified by the len field.

## 3.18 Get the IP Address Information Structure

### 3.18.1 Function Documentation

#### 3.18.1.1 `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 )`

Queries the IP address information on a link corresponding to the IP family provided (network order).

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srvc_Hndl()</a> .
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\\_Srvc\\_Hndl\(\)](#).

The length parameter can be obtained by calling [qapi\\_DSS\\_Get\\_IP\\_Addr\\_Count\(\)](#).

It is assumed that the client has allocated memory for enough structures specified by the len field.

## 3.19 Get the Link MTU Information

### 3.19.1 Function Documentation

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

Queries the MTU information associated with the link.

##### Parameters

in	<i>hndl</i>	Handle received from <a href="#">qapi_DSS_Get_Data_Srv_Hndl()</a> .
out	<i>mtu</i>	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\\_Srv\\_Hndl\(\)](#).

## 4 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 Socket Macros and Data Structures](#)
- [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](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 4.1 QAPI Socket Macros and Data Structures

This section provides the QAPI socket macros and data structures.

### BSD Socket Error Codes

- #define `ENOBUFFS` 1
- #define `ETIMEDOUT` 2
- #define `EISCONN` 3
- #define `EOPNOTSUPP` 4
- #define `ECONNABORTED` 5
- #define `EWOULDBLOCK` 6
- #define `ECONNREFUSED` 7
- #define `ECONNRESET` 8
- #define `ENOTCONN` 9
- #define `EBADF` 9
- #define `EALREADY` 10
- #define `EINVAL` 11
- #define `EMSGSIZE` 12
- #define `EPIPE` 13
- #define `EDESTADDRREQ` 14
- #define `ESHUTDOWN` 15
- #define `ENOPROTOOPT` 16
- #define `EHAVEOOB` 17
- #define `ENOMEM` 18
- #define `EADDRNOTAVAIL` 19
- #define `EADDRINUSE` 20
- #define `EAFNOSUPPORT` 21
- #define `EINPROGRESS` 22
- #define `ELOWER` 23
- #define `ENOTSOCK` 24
- #define `EIEIO` 27
- #define `ETOOMANYREFS` 28
- #define `EFAULT` 29
- #define `ENETUNREACH` 30

## 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\_TCPACK 0x00200
- #define SO\_WINSIZE 0x00400
- #define SO\_TIMESTAMP 0x00800
- #define SO\_BICWIND 0x01000
- #define SO\_HDRINCL 0x02000
- #define SO\_NOSLOWSTART 0x04000
- #define SO\_FULLMSS 0x08000
- #define SO\_SNDTIMEO 0x1005
- #define SO\_RCVTIMEO 0x1006
- #define SO\_ERROR 0x1007
- #define SO\_RXDATA 0x1011
- #define SO\_TXDATA 0x1012
- #define SO\_MYADDR 0x1013
- #define SO\_NBIO 0x1014
- #define SO\_BIO 0x1015
- #define SO\_NONBLOCK 0x1016
- #define SO\_CALLBACK 0x1017
- #define SO\_UDPCALLBACK 0x1019
- #define IPPROTO\_IP 0
- #define IP\_HDRINCL 2
- #define IP\_MULTICAST\_IF 9
- #define IP\_MULTICAST\_TTL 10



- #define `IP_MULTICAST_LOOP` 11
- #define `IP_ADD_MEMBERSHIP` 12
- #define `IP_DROP_MEMBERSHIP` 13
- #define `IPV6_MULTICAST_IF` 80
- #define `IPV6_MULTICAST_HOPS` 81
- #define `IPV6_MULTICAST_LOOP` 82
- #define `IPV6_JOIN_GROUP` 83
- #define `IPV6_LEAVE_GROUP` 84
- #define `IP_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

### 4.1.1 Define Documentation

#### 4.1.1.1 #define `AF_UNSPEC` 0

Address family unspecified.

#### 4.1.1.2 #define `AF_INET` 2

Address family IPv4.

#### 4.1.1.3 #define `AF_INET6` 3

Address family IPv6.

#### 4.1.1.4 #define `AF_INET_DUAL` 4

Address family IPv4 and IPv6.

**4.1.1.5 #define SOCK\_STREAM 1**

Socket stream (TCP).

**4.1.1.6 #define SOCK\_DGRAM 2**

Socket datagram (UDP).

**4.1.1.7 #define SOCK\_RAW 3**

Raw socket.

**4.1.1.8 #define ENOBUFS 1**

No buffer space is available.

**4.1.1.9 #define ETIMEDOUT 2**

Operation timed out.

**4.1.1.10 #define EISCONN 3**

Socket is already connected.

**4.1.1.11 #define EOPNOTSUPP 4**

Operation is not supported.

**4.1.1.12 #define ECONNABORTED 5**

Software caused a connection abort.

**4.1.1.13 #define EWOULDBLOCK 6**

Socket is marked nonblocking and the requested operation will block.

**4.1.1.14 #define ECONNREFUSED 7**

Connection was refused.

**4.1.1.15 #define ECONNRESET 8**

Connection was reset by peer.

**4.1.1.16 #define ENOTCONN 9**

Socket is not connected.

**4.1.1.17 #define EBADF 9**

An invalid descriptor was specified.

**4.1.1.18 #define EALREADY 10**

Operation is already in progress.

**4.1.1.19 #define EINVAL 11**

Invalid argument was passed.

**4.1.1.20 #define EMSGSIZE 12**

Message is too long.

**4.1.1.21 #define EPIPE 13**

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

**4.1.1.22 #define EDESTADDRREQ 14**

Destination address is required.

**4.1.1.23 #define ESHUTDOWN 15**

Cannot send after a socket shutdown.

**4.1.1.24 #define ENOPROTOPT 16**

Protocol is not available.

**4.1.1.25 #define EHAVEOOB 17**

Out of band.

**4.1.1.26 #define ENOMEM 18**

No memory is available.

**4.1.1.27 #define EADDRNOTAVAIL 19**

Cannot assign the requested address.

**4.1.1.28 #define EADDRINUSE 20**

Address is already in use.

**4.1.1.29 #define EAFNOSUPPORT 21**

Address family is not supported by the protocol family.

**4.1.1.30 #define EINPROGRESS 22**

Operation is in progress.

**4.1.1.31 #define ELOWER 23**

Lower layer (IP) error.

**4.1.1.32 #define ENOTSOCK 24**

Socket operation on nonsocket.

**4.1.1.33 #define EIEIO 27**

I/O error.

**4.1.1.34 #define ETOOMANYREFS 28**

Too many references.

**4.1.1.35 #define EFAULT 29**

Bad address.

**4.1.1.36 #define ENETUNREACH 30**

Network is unreachable.

**4.1.1.37 #define SOL\_SOCKET -1**

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

**4.1.1.38 #define SOL\_SOCKET -1**

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

**4.1.1.39 #define SO\_ACCEPTCONN 0x00002**

Socket has had listen().

**4.1.1.40 #define SO\_REUSEADDR 0x00004**

Allow local address reuse.

**4.1.1.41 #define SO\_KEEPALIVE 0x00008**

Keep connections alive.

**4.1.1.42 #define SO\_DONTROUTE 0x00010**

Not used.

**4.1.1.43 #define SO\_BROADCAST 0x00020**

Not used.

**4.1.1.44 #define SO\_USELOOPBACK 0x00040**

Not used.

**4.1.1.45 #define SO\_LINGER 0x00080**

Linger on close if data is present.

**4.1.1.46 #define SO\_OOBINLINE 0x00100**

Leave the received OOB data in line.

**4.1.1.47 #define SO\_Tcpsack 0x00200**

Allow TCP SACK (selective acknowledgment).

**4.1.1.48 #define SO\_WINSIZE 0x00400**

Set the scaling window option.

**4.1.1.49 #define SO\_TIMESTAMP 0x00800**

Set the TCP timestamp option.

**4.1.1.50 #define SO\_BICWND 0x01000**

Large initial TCP congestion window.

**4.1.1.51 #define SO\_HDRINCL 0x02000**

User access to IP header for SOCK\_RAW.

**4.1.1.52 #define SO\_NOSLOWSTART 0x04000**

Suppress slowstart on this socket.

**4.1.1.53 #define SO\_FULLMSS 0x08000**

Not used.

**4.1.1.54 #define SO\_SNDTIMEO 0x1005**

Send a timeout.

**4.1.1.55 #define SO\_RCVTIMEO 0x1006**

Receive a timeout.

**4.1.1.56 #define SO\_ERROR 0x1007**

Socket error.

**4.1.1.57 #define SO\_RXDATA 0x1011**

Get a count of bytes in sb\_rcv.

**4.1.1.58 #define SO\_TXDATA 0x1012**

Get a count of bytes in sb\_snd.

**4.1.1.59 #define SO\_MYADDR 0x1013**

Return my IP address.

**4.1.1.60 #define SO\_NBLOCK 0x1014**

Set socket to Nonblocking mode.

**4.1.1.61 #define SO\_BLOCK 0x1015**

Set socket to Blocking mode.

**4.1.1.62 #define SO\_NONBLOCK 0x1016**

Set/get blocking mode via the optval parameter.

**4.1.1.63 #define SO\_CALLBACK 0x1017**

Set/get the TCP zero\_copy callback routine.

**4.1.1.64 #define SO\_UDPCALLBACK 0x1019**

Set/get the UDP zero\_copy callback routine.

**4.1.1.65 #define IPPROTO\_IP 0**

For use with [gs]etsockopt() at IPPROTO\_IP level.

**4.1.1.66 #define IP\_HDRINCL 2**

IP header is included with the data.

**4.1.1.67 #define IP\_MULTICAST\_IF 9**

Set/get the IP multicast interface.

**4.1.1.68 #define IP\_MULTICAST\_TTL 10**

Set/get the IP multicast TTL.

**4.1.1.69 #define IP\_MULTICAST\_LOOP 11**

Set/get the IP multicast loopback.

**4.1.1.70 #define IP\_ADD\_MEMBERSHIP 12**

Add an IPv4 group membership.

**4.1.1.71 #define IP\_DROP\_MEMBERSHIP 13**

Drop an IPv4 group membership.

**4.1.1.72 #define IPV6\_MULTICAST\_IF 80**

Set the egress interface for multicast traffic.

**4.1.1.73 #define IPV6\_MULTICAST\_HOPS 81**

Set the number of hops.

**4.1.1.74 #define IPV6\_MULTICAST\_LOOP 82**

Enable/disable loopback for multicast.

**4.1.1.75 #define IPV6\_JOIN\_GROUP 83**

Join an IPv6 MC group.

**4.1.1.76 #define IPV6\_LEAVE\_GROUP 84**

Leave an IPv6 MC group.

**4.1.1.77 #define IP\_OPTIONS 1**

For use with [gs]setsockopt() at IP\_OPTIONS level.

**4.1.1.78 #define IP\_TOS 3**

IPv4 type of service and precedence.

**4.1.1.79 #define IP\_TTL\_OPT 4**

IPv4 time to live.

**4.1.1.80 #define IPV6\_SCOPEID 14**

IPv6 IF scope ID.

**4.1.1.81 #define IPV6\_UNICAST\_HOPS 15**

IPv6 hop limit.

**4.1.1.82 #define IPV6\_TCLASS 16**

IPv6 traffic class.

**4.1.1.83 #define MSG\_OOB 0x1**

Send/receive out-of-band data.

**4.1.1.84 #define MSG\_PEEK 0x2**

Peek at the incoming message.

**4.1.1.85 #define MSG\_DONTROUTE 0x4**

Send without using routing tables.

**4.1.1.86 #define MSG\_DONTWAIT 0x20**

Send/receive is nonblocking.

**4.1.1.87 #define MSG\_ZEROCOPYSEND 0x1000**

Send with zero-copy.

**4.1.1.88 #define QAPI\_NET\_WAIT\_FOREVER (0xFFFFFFFF)**

Infinite time for the timeout\_ms argument in [qapi\\_select\(\)](#).



**4.1.1.89 #define FD\_ZERO( set ) qapi\_fd\_zero((set))**

Clears a set.

**4.1.1.90 #define FD\_CLR( handle, set ) qapi\_fd\_clr((handle), (set))**

Removes a given file descriptor from a set.

**4.1.1.91 #define FD\_SET( handle, set ) qapi\_fd\_set((handle), (set))**

Adds a given file descriptor from a set.

**4.1.1.92 #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.

**4.1.2 Data Structure Documentation****4.1.2.1 struct in\_addr**

IPv4 Internet address.

**Data fields**

Type	Parameter	Description
uint32_t	s_addr	IPv4 address in network order.

**4.1.2.2 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.

**4.1.2.3 struct in6\_addr**

IPv6 Internet address.

**Data fields**

Type	Parameter	Description
uint8_t	s_addr	128-bit IPv6 address.

#### 4.1.2.4 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.

#### 4.1.2.5 union ip46addr\_n.a

##### Data fields

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

#### 4.1.2.6 union ip46addr\_n.g

##### Data fields

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

#### 4.1.2.7 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.

#### 4.1.2.8 struct ip46addr

Socket IPv4/IPv6 Internet address union.

##### Data fields

Type	Parameter	Description
uint16_t	type	AF_INET or AF_INET6.
union <a href="#">ip46addr</a>	a	Address union.

#### 4.1.2.9 union ip46addr.a

##### Data fields

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

#### 4.1.2.10 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.

#### 4.1.2.11 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.

## 4.2 Create a Socket

### 4.2.1 Function Documentation

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

Creates an endpoint for communication.

##### Parameters

in	<i>family</i>	Protocol family used for communication. The supported families are: <ul style="list-style-type: none"><li>• AF_INET – IPv4 Internet protocols</li><li>• AF_INET6 – IPv6 Internet protocols</li></ul>
in	<i>type</i>	Transport mechanism used for communication. The supported types are: <ul style="list-style-type: none"><li>• SOCK_STREAM – TCP</li><li>• SOCK_DGRAM – UDP</li></ul>
in	<i>protocol</i>	Must be set to 0.

##### Returns

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

On error, -1 is returned.

## 4.3 Bind a Socket

### 4.3.1 Function Documentation

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

Assigns an address to the socket created by [qapi\\_socket\(\)](#).

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
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 is returned. On error, -1 is returned.

## 4.4 Make a Socket Passive

### 4.4.1 Function Documentation

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

Marks the socket as a passive socket.

##### Parameters

in	<i>handle</i>	Handle (returned from <a href="#">qapi_socket()</a> ) 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 is returned. On error, -1 is returned.

## 4.5 Accept a Socket Connection Request

### 4.5.1 Function Documentation

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

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.

#### Parameters

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

## 4.6 Connect to a Socket

### 4.6.1 Function Documentation

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

Initiates a connection on a socket

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

##### Parameters

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

##### Returns

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



## 4.7 Set Socket Options

### 4.7.1 Function Documentation

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

Sets the options for a socket.

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
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 is returned. On error, -1 is returned.

## 4.8 Get Socket Options

### 4.8.1 Function Documentation

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

Gets the options for a socket.

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
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 is returned. On error, -1 is returned.

## 4.9 Close a Socket

### 4.9.1 Function Documentation

#### 4.9.1.1 `qapi_Status_t qapi_socketclose ( int32_t handle )`

Closes a socket.

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
----	---------------	---

##### Returns

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

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 4.10 Get a Socket Error Code

### 4.10.1 Function Documentation

#### 4.10.1.1 `int qapi_errno ( int32_t handle )`

Gets the last error code on a socket.

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
----	---------------	---

##### Returns

Socket error code or ENOTSOCK if socket is not found

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 4.11 Receive a Message from a Socket

### 4.11.1 Function Documentation

#### 4.11.1.1 `int qapi_rcvfrom ( int32_t handle, char * buf, int32_t len, int32_t flags, struct sockaddr * from, int32_t * fromlen )`

Receives a message from a socket.

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
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: <ul style="list-style-type: none"> <li>• 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.</li> <li>• MSG_OOB – Requests receipt of out-of-band data that would not be received in the normal data stream.</li> <li>• MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.</li> </ul>
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, <i>fromlen</i> 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 <i>from</i> , 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.

## 4.12 Receive a Message from a Connected Socket

### 4.12.1 Function Documentation

#### 4.12.1.1 `int qapi_rcv ( int32_t handle, char * buf, int32_t len, int32_t flags )`

Receives a message from a socket.

The `qapi_rcv()` call is normally used only on a connected socket and is identical to `qapi_rcvfrom(handle, buf, len, flags, NULL, NULL)`

##### Parameters

in	<i>handle</i>	Socket handle returned from <code>qapi_socket()</code> .
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: <ul style="list-style-type: none"> <li>• 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.</li> <li>• MSG_OOB – Requests receipt of out-of-band data that would not be received in the normal data stream.</li> <li>• MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.</li> </ul>

##### Returns

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

## 4.13 Send a Message on a Socket

### 4.13.1 Function Documentation

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

Sends a message on a socket to a target.

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
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: <ul style="list-style-type: none"> <li>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.</li> <li>MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.</li> <li>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.</li> </ul>
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 errno is set appropriately.

## 4.14 Send a Message on a Connected Socket

### 4.14.1 Function Documentation

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

Send 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)`

##### Parameters

in	<i>handle</i>	Socket handle returned from <a href="#">qapi_socket()</a> .
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: <ul style="list-style-type: none"> <li>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.</li> <li>MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.</li> <li>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.</li> </ul>

##### Returns

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



## 4.15 Select a Socket

### 4.15.1 Function Documentation

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

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. The call only supports read socket set, so "*wr*" and "*ex*" must be set to NULL.

#### Parameters

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

#### Returns

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

## 4.16 Initialize a Socket

### 4.16.1 Function Documentation

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

Initializes a socket that is set to zero.

##### Parameters

<code>in</code>	<code>set</code>	Pointer to a list of sockets.
-----------------	------------------	-------------------------------

##### Returns

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

## 4.17 Clear a Socket from a Socket Set

### 4.17.1 Function Documentation

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

Removes a socket from the socket set.

##### Parameters

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

##### Returns

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

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

## 4.18 Add a Socket to a Socket Set

### 4.18.1 Function Documentation

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

Adds a socket to the socket set.

##### Parameters

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

##### Returns

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

## 4.19 Check Whether a Socket is in a Socket Set

### 4.19.1 Function Documentation

#### 4.19.1.1 `qapi_Status_t qapi_fd_isset ( int32_t handle, fd_set * set )`

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

##### Parameters

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

##### Returns

On success, 0 is returned if the socket is not a member; 1 is returned if the socket is a member.

On error, -1 is returned.

## 4.20 Get the Address of a Connected Peer

### 4.20.1 Function Documentation

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

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

##### Parameters

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

##### Returns

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

## 4.21 Get the Address to Which the Socket is Bound

### 4.21.1 Function Documentation

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

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

##### Parameters

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

##### Returns

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

# 5 QAPI Network Security APIs

---

This chapter describes the QAPIs used for transport layer security (TLS) and datagram transport layer security (DTLS).

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](#)
- [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](#)



## 5.1 QAPI SSL Data Types

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

### 5.1.1 Define Documentation

#### 5.1.1.1 **#define QAPI\_NET\_SSL\_MAX\_CERT\_NAME\_LEN (32)**

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

#### 5.1.1.2 **#define QAPI\_NET\_SSL\_MAX\_NUM\_CERTS (10)**

Maximum number of file names returned in the [qapi\\_Net\\_SSL\\_Cert\\_List\(\)](#) API.

#### 5.1.1.3 **#define QAPI\_NET\_SSL\_CIPHERSUITE\_LIST\_DEPTH 8**

Maximum number of cipher suites that can be configured.

#### 5.1.1.4 **#define QAPI\_NET\_SSL\_INVALID\_HANDLE (0)**

Invalid handle.

#### 5.1.1.5 **#define QAPI\_NET\_SSL\_PROTOCOL\_UNKNOWN 0x00**

Unknown SSL protocol version.

#### 5.1.1.6 **#define QAPI\_NET\_SSL\_PROTOCOL\_TLS\_1\_0 0x31**

TLS version 1.0.

#### 5.1.1.7 **#define QAPI\_NET\_SSL\_PROTOCOL\_TLS\_1\_1 0x32**

TLS version 1.1.

#### 5.1.1.8 **#define QAPI\_NET\_SSL\_PROTOCOL\_TLS\_1\_2 0x33**

TLS version 1.2.

#### 5.1.1.9 **#define QAPI\_NET\_SSL\_PROTOCOL\_DTLS\_1\_0 0xEF**

DTLS version 1.0.

#### 5.1.1.10 **#define QAPI\_NET\_SSL\_PROTOCOL\_DTLS\_1\_2 0xED**

DTLS version 1.2.

**5.1.1.11 #define QAPI\_NET\_TLS\_PSK\_WITH\_RC4\_128\_SHA 0x008A**

TLS PSK with RC4 128 SHA.

**5.1.1.12 #define QAPI\_NET\_TLS\_PSK\_WITH\_3DES\_EDE\_CBC\_SHA 0x008B**

TLS PSK with 3DES EDE CBC SHA.

**5.1.1.13 #define QAPI\_NET\_TLS\_PSK\_WITH\_AES\_128\_CBC\_SHA 0x008C**

TLS PSK with AES 128 CBC SHA.

**5.1.1.14 #define QAPI\_NET\_TLS\_PSK\_WITH\_AES\_256\_CBC\_SHA 0x008D**

TLS PSK with AES 256 CBC SHA.

**5.1.1.15 #define QAPI\_NET\_TLS\_PSK\_WITH\_AES\_128\_GCM\_SHA256 0x00A8**

TLS PSK with AES\_128 GCM SHA256.

**5.1.1.16 #define QAPI\_NET\_TLS\_PSK\_WITH\_AES\_256\_GCM\_SHA384 0x00A9**

TLS PSK with AES 256 GCM SHA384.

**5.1.1.17 #define QAPI\_NET\_TLS\_PSK\_WITH\_AES\_128\_CBC\_SHA256 0x00AE**

TLS PSK with AES 128 CBC SHA256.

**5.1.1.18 #define QAPI\_NET\_TLS\_PSK\_WITH\_AES\_256\_CBC\_SHA384 0x00AF**

TLS PSK with AES 256 CBC SHA384.

**5.1.1.19 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA 0x002F**

Cipher TLS RSA with AES 128 CBC SHA.

**5.1.1.20 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA 0x0033**

Cipher TLS DHE RSA with AES 128 CBC SHA.

**5.1.1.21 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA 0x0035**

Cipher TLS RSA with AES 256 CBC SHA.

**5.1.1.22 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA 0x0039**

Cipher TLS DHE RSA with AES 256 CBC SHA.

**5.1.1.23 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256 0x003C**

Cipher TLS RSA with AES 128 CBC SHA256.

**5.1.1.24 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256 0x003D**

Cipher TLS RSA with AES 256 CBC SHA256.

**5.1.1.25 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 0x0067**

Cipher TLS DHE RSA with AES 128 CBC SHA256.

**5.1.1.26 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_256\_CBC\_SHA256 0x006B**

Cipher TLS DHE RSA with AES 256 CBC SHA256.

**5.1.1.27 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 0x009C**

Cipher TLS RSA with AES 128 GCM SHA256.

**5.1.1.28 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384 0x009D**

Cipher TLS RSA with AES 256 GCM SHA384.

**5.1.1.29 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 0x009E**

Cipher TLS DHE RSA with AES 128 GCM SHA256.

**5.1.1.30 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 0x009F**

Cipher TLS DHE RSA with AES 256 GCM SHA384.

**5.1.1.31 #define QAPI\_NET\_TLS\_ECDH\_ECDSA\_WITH\_AES\_128\_CBC\_SHA 0xC004**

Cipher TLS ECDH ECDSA with AES 128 CBC SHA.

**5.1.1.32 #define QAPI\_NET\_TLS\_ECDH\_ECDSA\_WITH\_AES\_256\_CBC\_SHA 0xC005**

Cipher TLS ECDH ECDSA with AES 256 CBC SHA.

**5.1.1.33 #define QAPI\_NET\_TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA 0xC009**

Cipher TLS ECDHE ECDSA with AES 128 CBC SHA.

**5.1.1.34 #define QAPI\_NET\_TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA 0xC00A**

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA.

**5.1.1.35 #define QAPI\_NET\_TLS\_ECDH\_RSA\_WITH\_AES\_128\_CBC\_SHA 0xC00E**

Cipher TLS ECDH RSA with AES 128 CBC SHA.

**5.1.1.36 #define QAPI\_NET\_TLS\_ECDH\_RSA\_WITH\_AES\_256\_CBC\_SHA 0xC00F**

Cipher TLS ECDH RSA with AES 256 CBC SHA.

**5.1.1.37 #define QAPI\_NET\_TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA 0xC013**

Cipher TLS ECDHE RSA with AES 128 CBC SHA.

**5.1.1.38 #define QAPI\_NET\_TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA 0xC014**

Cipher TLS ECDHE RSA with AES 256 CBC SHA.

**5.1.1.39 #define QAPI\_NET\_TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256 0xC023**

Cipher TLS ECDHE ECDSA with AES 128 CBC SHA256.

**5.1.1.40 #define QAPI\_NET\_TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384 0xC024**

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA384.

**5.1.1.41 #define QAPI\_NET\_TLS\_ECDH\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256 0xC025**

Cipher TLS ECDH ECDSA with AES 128 CBC SHA256.

**5.1.1.42 #define QAPI\_NET\_TLS\_ECDH\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384 0xC026**

Cipher TLS ECDH ECDSA with AES 256 CBC SHA384.

**5.1.1.43 #define QAPI\_NET\_TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256 0xC027**

Cipher TLS ECDHE RSA with AES 128 CBC SHA256.

**5.1.1.44 #define QAPI\_NET\_TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384 0xC028**

Cipher TLS ECDHE RSA with AES 256 CBC SHA384.

**5.1.1.45 #define QAPI\_NET\_TLS\_ECDH\_RSA\_WITH\_AES\_128\_CBC\_SHA256 0xC029**

Cipher TLS ECDH RSA with AES 128 CBC SHA256.

**5.1.1.46 #define QAPI\_NET\_TLS\_ECDH\_RSA\_WITH\_AES\_256\_CBC\_SHA384 0xC02A**

Cipher TLS ECDH RSA with AES 256 CBC SHA384.

**5.1.1.47 #define QAPI\_NET\_TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 0xC02B**

Cipher TLS ECDHE ECDSA with AES 128 GCM SHA256.

**5.1.1.48 #define QAPI\_NET\_TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384 0xC02C**

Cipher TLS ECDHE ECDSA with AES 256 GCM SHA384.

**5.1.1.49 #define QAPI\_NET\_TLS\_ECDH\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 0xC02D**

Cipher TLS ECDH ECDSA with AES 128 GCM SHA256.

**5.1.1.50 #define QAPI\_NET\_TLS\_ECDH\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384 0xC02E**

Cipher TLS ECDH ECDSA with AES 256 GCM SHA384.

**5.1.1.51 #define QAPI\_NET\_TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256 0xC02F**

Cipher TLS ECDHE RSA with AES 128 GCM SHA256.

**5.1.1.52 #define QAPI\_NET\_TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384 0xC030**

Cipher TLS ECDHE RSA with AES 256 GCM SHA384.

**5.1.1.53 #define QAPI\_NET\_TLS\_ECDH\_RSA\_WITH\_AES\_128\_GCM\_SHA256 0xC031**

Cipher TLS ECDH RSA with AES 128 GCM SHA256.

**5.1.1.54 #define QAPI\_NET\_TLS\_ECDH\_RSA\_WITH\_AES\_256\_GCM\_SHA384 0xC032**

Cipher TLS ECDH RSA with AES 256 GCM SHA384.

**5.1.1.55 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_128\_CCM 0xC09C**

Cipher TLS RSA with AES 128 CCM.

**5.1.1.56 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_256\_CCM 0xC09D**

Cipher TLS RSA with AES 256 CCM.

**5.1.1.57 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_128\_CCM 0xC09E**

Cipher TLS DHE RSA with AES 128 CCM.

**5.1.1.58 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_256\_CCM 0xC09F**

Cipher TLS DHE RSA with AES 256 CCM.

**5.1.1.59 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_128\_CCM\_8 0xC0A0**

Cipher TLS RSA with AES 128 CCM 8.

**5.1.1.60 #define QAPI\_NET\_TLS\_RSA\_WITH\_AES\_256\_CCM\_8 0xC0A1**

Cipher TLS RSA with AES 256 CCM 8.

**5.1.1.61 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_128\_CCM\_8 0xC0A2**

Cipher TLS DHE RSA with AES 128 CCM 8.

**5.1.1.62 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_AES\_256\_CCM\_8 0xC0A3**

Cipher TLS DHE RSA with AES 256 CCM 8.

**5.1.1.63 #define QAPI\_NET\_TLS\_ECDHE\_RSA\_WITH\_CHACHA20\_POLY1305\_SHA256 0xCC13**

Cipher TLS ECDHE RSA with CHACHA20 POLY1305 SHA256.

**5.1.1.64 #define QAPI\_NET\_TLS\_ECDHE\_ECDSA\_WITH\_CHACHA20\_POLY1305\_SHA256 0xCC14**

Cipher TLS ECDHE ECDSA with CHACHA20 POLY1305 SHA256.

**5.1.1.65 #define QAPI\_NET\_TLS\_DHE\_RSA\_WITH\_CHACHA20\_POLY1305\_SHA256 0xCC15**

Cipher TLS DHE RSA with CHACHA20 POLY1305 SHA256.

**5.1.1.66 #define QAPI\_NET\_SSL\_MAX\_CA\_LIST 10**

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

**5.1.2 Data Structure Documentation****5.1.2.1 struct \_\_qapi\_Net\_SSL\_Verify\_Policy\_s**

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_Velocity	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 <a href="#">QAPI_NET_SSL_MAX_CERT_NAME_LEN</a> .

**5.1.2.2 struct \_\_qapi\_Net\_SSL\_Config\_s**

Structure to configure an SSL connection.

**Data fields**

Type	Parameter	Description
uint16_t	protocol	Protocol to use. See <a href="#">QAPI_NET_SSL_PROTOCOL_*</a> .
uint16_t	cipher	Cipher to use. See SSL cipher suites <a href="#">QAPI_NET_TLS*</a> and <a href="#">QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH</a> .
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.

**5.1.2.3 struct \_\_qapi\_Net\_SSL\_Cert\_List\_s**

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

**Data fields**

Type	Parameter	Description
char	name	Certificate name. See <a href="#">QAPI_NET_SSL_MAX_NUM_CERTS</a> and <a href="#">QAPI_NET_SSL_MAX_CERT_NAME_LEN</a> .

**5.1.2.4 struct \_\_qapi\_Net\_SSL\_CERT\_s**

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.

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

### 5.1.2.5 struct \_\_qapi\_NET\_SSL\_CA\_Info\_s

SSL certificate authority list information.

#### Data fields

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

### 5.1.2.6 struct \_\_qapi\_Net\_SSL\_CA\_List\_s

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.

### 5.1.2.7 struct \_\_qapi\_Net\_SSL\_PSK\_Table\_s

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.

### 5.1.2.8 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.



Type	Parameter	Description
union __qapi_Net_SSL_Cert_Info_s	info	Certificate information.

### 5.1.2.9 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.

## 5.1.3 Enumeration Type Documentation

### 5.1.3.1 enum qapi\_Net\_SSL\_Role\_t

SSL object role.

#### Enumerator:

**QAPI\_NET\_SSL\_SERVER\_E** Server role.  
**QAPI\_NET\_SSL\_CLIENT\_E** Client role.

### 5.1.3.2 enum qapi\_Net\_SSL\_Protocol\_t

SSL protocol.

#### Enumerator:

**QAPI\_NET\_SSL\_TLS\_E** TLS protocol.  
**QAPI\_NET\_SSL\_DTLS\_E** DTLS protocol.

### 5.1.3.3 enum qapi\_Net\_SSL\_Cert\_Type\_t

SSL certificate type.

#### Enumerator:

**QAPI\_NET\_SSL\_CERTIFICATE\_E** Certificate type.  
**QAPI\_NET\_SSL\_CA\_LIST\_E** CA list type  
**QAPI\_NET\_SSL\_PSK\_TABLE\_E** PSK key table type.

## 5.2 QAPI SSL Typedefs

This section provides the typedefs for the networking SSL.

### 5.2.1 Typedef Documentation

#### 5.2.1.1 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\(\)](#).

#### 5.2.1.2 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\(\)](#).

#### 5.2.1.3 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>]
```

#### 5.2.1.4 typedef const void\* qapi\_Net\_SSL\_CAList\_t

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

##### Usage

```
SharkSSLParseCAList [-b <binary output file>] <certfile> [certfile...]
where certfile is a .PEM, .DER or .P7B file containing one or more certificates
```

#### 5.2.1.5 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>]
```

## 5.3 Create an SSL Object

### 5.3.1 Function Documentation

#### 5.3.1.1 `qapi_Net_SSL_Obj_Hdl_t qapi_Net_SSL_Obj_New ( qapi_Net_SSL_Role_t role )`

Creates a new SSL object (server or client).

##### Parameters

in	<i>role</i>	Server or client role.
----	-------------	------------------------

##### 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.

## 5.4 Create an SSL Connection Handle

### 5.4.1 Function Documentation

#### 5.4.1.1 `qapi_Net_SSL_Con_Hdl_t qapi_Net_SSL_Con_New ( qapi_Net_SSL_Obj_Hdl_t hdl, qapi_Net_SSL_Protocol_t prot )`

Creates an SSL connection handle for an SSL object.

##### 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).

## 5.5 Configure an SSL Connection

### 5.5.1 Function Documentation

#### 5.5.1.1 `qapi_Status_t qapi_Net_SSL_Configure ( qapi_Net_SSL_Con_Hdl_t ssl, qapi_Net_SSL_Config_t * cfg )`

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

The SSL protocol and up to 8 ciphers can be configured in the client context.

The `SSL_VERIFY_POLICY` verify structure (and `matchName`) specify how the SSL certificate will be verified during the SSL handshake:

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

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

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

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

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

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

There can be scenarios where the peer does not support the maximum fragment length TLS extension, but the maximum fragment length is inferred. In that case, the user may choose to configure `max_frag_len` 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`. Then the above limitations do not apply.

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

#### Parameters

<code>in</code>	<code>ssl</code>	Connection handle.
-----------------	------------------	--------------------

<i>in</i>	<i>cfg</i>	Configuration parameters.
-----------	------------	---------------------------

**Returns**

QAPI\_OK on success.

QAPI\_ERR\_INVALID\_PARAM\_SSL if an error occurred (configuration is invalid).

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 5.6 Delete an SSL Certificate

### 5.6.1 Function Documentation

#### 5.6.1.1 `qapi_Status_t qapi_Net_SSL_Cert_delete ( char * name, qapi_Net_SSL_Cert_Type_t type )`

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

##### 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.

## 5.7 Store an SSL Certificate

### 5.7.1 Function Documentation

#### 5.7.1.1 `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 )`

Stores an internal certificate, 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

#### 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.



## 5.8 Convert and Store an SSL Certificate

### 5.8.1 Function Documentation

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

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

The maximum number of CA lists that are supported for conversion and storage in binary format is QAPI\_NET\_SSL\_MAX\_CA\_LIST.

The PSK table must be in the following format:

- Identity\_1: psk\_key1
- Identity\_2: psk\_key2

#### Parameters

in	<i>cert_info</i>	Information pertaining to either the client certificate, CA lists in .PEM, .DER, or .P7B format, or PSK tables.
in	<i>cert_name</i>	Name of the certificate, CA list, or PSK table that the <i>cert_info</i> is to be stored under after the conversion.

#### Returns

0 on success.  
Negative value on error.

## 5.9 Load an SSL Certificate

### 5.9.1 Function Documentation

#### 5.9.1.1 `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 )`

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

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

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

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

Certificates, CA lists, or a PSK table must be added before the [qapi\\_Net\\_SSL\\_Connect\(\)](#) or [qapi\\_Net\\_SSL\\_Accept\(\)](#) APIs are called.

#### 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.

## 5.10 Get a List of SSL Certificates

### 5.10.1 Function Documentation

#### 5.10.1.1 `qapi_Status_t qapi_Net_SSL_Cert_List ( qapi_Net_SSL_Cert_Type_t type, qapi_Net_SSL_Cert_List_t * list )`

Gets a list of encrypted certificates, CA lists, or a PSK tables stored in nonvolatile memory.

The structure `__qapi_Net_SSL_Cert_List_s` must be allocated by the caller.

##### 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.

## 5.11 Attach a Socket Descriptor to the SSL Connection

### 5.11.1 Function Documentation

#### 5.11.1.1 `qapi_Status_t qapi_Net_SSL_Fd_Set ( qapi_Net_SSL_Con_Hdl_t ssl, uint32_t fd )`

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.).

#### 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.

## 5.12 Accept an SSL Connection From the Client

### 5.12.1 Function Documentation

#### 5.12.1.1 `qapi_Status_t qapi_Net_SSL_Accept ( qapi_Net_SSL_Con_Hdl_t ssl )`

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.

##### Parameters

in	<i>ssl</i>	SSL connection handle.
----	------------	------------------------

##### Returns

QAPI\_SSL\_OK\_HS on success.

QAPI\_ERR\_\* on error.

## 5.13 Initiate an SSL Handshake

### 5.13.1 Function Documentation

#### 5.13.1.1 `qapi_Status_t qapi_Net_SSL_Connect ( qapi_Net_SSL_Con_Hdl_t ssl )`

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

##### Parameters

in	<i>ssl</i>	SSL connection handle.
----	------------	------------------------

##### Returns

QAPI\_SSL\_OK\_HS on success.

QAPI\_ERR\_\* on error.

## 5.14 Close an SSL Connection

### 5.14.1 Function Documentation

#### 5.14.1.1 `qapi_Status_t qapi_Net_SSL_Shutdown ( qapi_Net_SSL_Con_Hdl_t ssl )`

Closes an SSL connection.

The connection handle will be freed in this API. The socket must be closed explicitly after this call. See [qapi\\_socketclose\(\)](#).

##### Parameters

in	<i>ssl</i>	SSL connection handle.
----	------------	------------------------

##### Returns

QAPI\_OK on success.

QAPI\_ERR\_INVALID\_PARAM\_SSL on error (invalid connection handle).

## 5.15 Free an SSL Object Handle

### 5.15.1 Function Documentation

#### 5.15.1.1 `qapi_Status_t qapi_Net_SSL_Obj_Free ( qapi_Net_SSL_Obj_Hdl_t hdl )`

Frees the SSL object handle.

##### 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.



## 5.16 Read SSL Data

### 5.16.1 Function Documentation

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

Reads data received over the SSL connection.

##### 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.

## 5.17 Write SSL Data

### 5.17.1 Function Documentation

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

Sends data over the SSL connection.

##### 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.

## 6 QAPI Networking Services

---

This chapter describes the Networking Services and utilities QAPIs.

- [Networking Services Macros, Data Types, and Enumerations](#)
- [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 Resonse](#)
- [Get the IPv6 Address of an Interface](#)
- [IPv6 Route Commands](#)
- [Get the Interface Scope ID](#)

## 6.1 Networking Services Macros, Data Types, and Enumerations

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

### 6.1.1 Define Documentation

#### 6.1.1.1 **#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	<i>ipv4_Address</i>	IPv4 address to check; must be in host order.
----	---------------------	---

##### Returns

1 if the IPv4 address is multicast, 0 otherwise.

#### 6.1.1.2 **#define IF\_NAMELEN 20**

Default maximum length for interface names.

#### 6.1.1.3 **#define QAPI\_NET\_IPV4\_MAX\_ROUTES (3)**

Maximum IPv4 routing configurations.

#### 6.1.1.4 **#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, or 0 otherwise.

**6.1.1.5 #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.

**6.1.1.6 #define QAPI\_NET\_IPV6\_MAX\_ROUTES (3)**

Maximum IPv6 routing configurations.

**6.1.1.7 #define QAPI\_NET\_IFNAME\_LEN 12**

Maximum length for the interface name.

**6.1.2 Data Structure Documentation****6.1.2.1 struct qapi\_Net\_Ping\_V4\_s**

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).

**6.1.2.2 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.

Type	Parameter	Description
char	ifName	Textual name of the interface.

### 6.1.2.3 struct qapi\_Net\_IPv4\_Route\_List\_t

IPv4 routing objects list.

#### Data fields

Type	Parameter	Description
uint32_t	route_Count	Number of <a href="#">qapi_Net_IPv4_Route_t</a> arrays in the routing table.
<a href="#">qapi_Net_IPv4_Route_t</a>	route	Array of <a href="#">qapi_Net_IPv4_Route_t</a> types.

### 6.1.2.4 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.

### 6.1.2.5 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	ipv6RouteIf-Index	Index value that uniquely identifies the local interface through which the next hop of this route should be reached.
char	ifName	Textual name of the interface.

### 6.1.2.6 struct qapi\_Net\_IPv6\_Route\_List\_t

IPv6 routing objects list.

#### Data fields

Type	Parameter	Description
uint32_t	route_Count	Number of <a href="#">qapi_Net_IPv6_Route_t</a> arrays in the routing table.
<a href="#">qapi_Net_IPv6_Route_t</a>	route	Array of type <a href="#">qapi_Net_IPv6_Route_t</a> .

### 6.1.2.7 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 <a href="#">qapi_Net_Get_Number_of_Interfaces()</a> if the value is $\geq 1$ .
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).

### 6.1.2.8 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.

## 6.1.3 Enumeration Type Documentation

### 6.1.3.1 enum qapi\_Net\_Route\_Command\_t

Commands for routing net services.

#### Enumerator:

**QAPI\_NET\_ROUTE\_ADD\_E** Add route.  
**QAPI\_NET\_ROUTE\_DEL\_E** Delete route.  
**QAPI\_NET\_ROUTE\_SHOW\_E** Show routes.

### 6.1.3.2 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).

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@quectel.com



## 6.2 Get the Names of All Network Interfaces

### 6.2.1 Function Documentation

#### 6.2.1.1 `int32_t qapi_Net_Get_All_Ifnames ( qapi_Net_Ifnameindex_t * ifname_index )`

Retrieves the textual names of all network interfaces.

##### Parameters

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

##### Returns

Number of network interfaces

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 6.3 Parse an Address String into an IPv4/IPv6 Address

### 6.3.1 Function Documentation

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

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

##### 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.

## 6.4 Format an IPv4/IPv6 Address into a NULL-terminated String

### 6.4.1 Function Documentation

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

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

##### 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, else NULL.

## 6.5 Get the Physical Address and Length of an Interface

### 6.5.1 Function Documentation

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

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

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

```
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
}
```

#### 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.

## 6.6 Check Whether an Interface Exists

### 6.6.1 Function Documentation

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

Checks whether the interface exists.

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

#### 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.

## 6.7 IPv4 Network Configuration

### 6.7.1 Function Documentation

#### 6.7.1.1 `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 )`

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

##### Parameters

in	<i>interface_Name</i>	Pointer to the interface name.
in	<i>cmd</i>	Command mode. Possible values are: <ul style="list-style-type: none"> <li>QAPI_NET_IPv4CFG_QUERY_E (0) – Get the IPv4 parameters of an interface.</li> <li>QAPI_NET_IPv4CFG_STATIC_IP_E (1) – Assign the IPv4 address, subnet mask, and default gateway.</li> </ul>
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.

## 6.8 Send an IPv4 Ping

### 6.8.1 Function Documentation

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

Sends an IPv4 ping.

##### 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.

## 6.9 Send an IPv4 Ping with a Response

### 6.9.1 Function Documentation

#### 6.9.1.1 `qapi_Status_t qapi_Net_Ping_2 ( qapi_Net_Ping_V4_t * ping_buf, qapi_Ping_Info_Resp_t * ping_resp )`

Sends an IPv4 ping request.

##### 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.



## 6.10 IPv4 Route Commands

### 6.10.1 Function Documentation

**6.10.1.1** `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 )`

Adds, deletes, or queries the IPv4 route.

#### Parameters

in	<i>interface_Name</i>	Pointer to the interface name.
in	<i>cmd</i>	Command mode. Possible values are: <ul style="list-style-type: none"> <li>QAPI_NET_ROUTE_ADD_E (0) – Add route.</li> <li>QAPI_NET_ROUTE_DEL_E (1) – Delete route.</li> <li>QAPI_NET_ROUTE_SHOW_E (2) – Show route.</li> </ul>
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.

## 6.11 Send an IPv6 Ping

### 6.11.1 Function Documentation

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

Sends an IPv6 ping request.

##### 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.

## 6.12 Send an IPv6 Ping with a Resonse

### 6.12.1 Function Documentation

#### 6.12.1.1 `qapi_Status_t qapi_Net_Ping6_2 ( qapi_Net_Ping_V6_t * ping6_buf, qapi_Ping_Info_Resp_t * ping_resp )`

Sends an IPv6 ping request with a response.

##### 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.

## 6.13 Get the IPv6 Address of an Interface

### 6.13.1 Function Documentation

**6.13.1.1** `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 )`

Gets the IPv6 addresses of an interface.

#### 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.

## 6.14 IPv6 Route Commands

### 6.14.1 Function Documentation

**6.14.1.1** `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 )`

Adds, deletes, or queries the IPv6 route.

#### Parameters

in	<i>interface_Name</i>	Pointer to the name of the network interface.
in	<i>cmd</i>	Command mode. Possible values are: <ul style="list-style-type: none"> <li>QAPI_NET_ROUTE_ADD_E (0) – Add route</li> <li>QAPI_NET_ROUTE_DEL_E (1) – Delete route</li> <li>QAPI_NET_ROUTE_SHOW_E (2) – Show route</li> </ul>
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.

## 6.15 Get the Interface Scope ID

### 6.15.1 Function Documentation

#### 6.15.1.1 `qapi_Status_t qapi_Net_IPv6_Get_Scope_ID ( const char * interface_Name, int32_t * 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.

##### 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.

# 7 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](#)

## 7.1 DNS Client Service Macros, Data Types, and Enumerations

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

### 7.1.1 Define Documentation

#### 7.1.1.1 #define MAX\_DNS\_SVR\_NUM 4

For use with [qapi\\_Net\\_DNSc\\_Get\\_Server\\_List\(\)](#) to get IP addresses of DNS servers.

#### 7.1.1.2 #define QAPI\_DNS\_PORT 53

DNS server port.

#### 7.1.1.3 #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.

#### 7.1.1.4 #define gethostbyname( \_\_name ) qapi\_Net\_DNSc\_Get\_Host\_By\_Name(\_\_name)

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

### 7.1.2 Data Structure Documentation

#### 7.1.2.1 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 <a href="#">ip46addr</a>	svr	DNS servers IP addresses.

#### 7.1.2.2 struct qapi\_hostent\_s

Data structure returned from [qapi\\_Net\\_DNSc\\_Get\\_Host\\_By\\_Name\(\)](#) or [qapi\\_Net\\_DNSc\\_Get\\_Host\\_By\\_Name2\(\)](#). 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.



## 7.1.3 Enumeration Type Documentation

### 7.1.3.1 enum qapi\_Net\_DNS\_Command\_t

Commands to start/stop/disable a DNS client.

**Enumerator:**

**QAPI\_NET\_DNS\_DISABLE\_E** Stop plus free the space for internal data structures.

**QAPI\_NET\_DNS\_START\_E** Allocate space for internal data structures; DNS query is allowed after the start command. DNS responses from the server.

**QAPI\_NET\_DNS\_STOP\_E** Stop sending DNS requests and processing DNS responses; keep internal data structures.

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

## 7.2 Check Whether the DNS Client has Started

### 7.2.1 Function Documentation

#### 7.2.1.1 `int32_t qapi_Net_DNSc_Is_Started ( void )`

Checks whether the DNS client has started.

##### Returns

0 if not started or 1 if started.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 7.3 Start, Stop, or Disable the DNS Client

### 7.3.1 Function Documentation

#### 7.3.1.1 `int32_t qapi_Net_DNSc_Command ( qapi_Net_DNS_Command_t cmd )`

Starts, stops, or disables the DNS client.

##### Parameters

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

##### Returns

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

## 7.4 Convert an IP Address Text String into an IP Address

### 7.4.1 Function Documentation

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

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

##### Parameters

in	<i>hostname</i>	Pointer to an IP address string or host name string.
in	<i>ipaddr</i>	Pointer to struct <code>ip46addr</code> for the resolved IP address. The caller must specify which IP address (v4 or v6) it intends to resolve to: If <code>ipaddr</code> type is <code>AF_INET</code> , resolve to an IPv4 address. If <code>ipaddr</code> type is <code>AF_INET6</code> , resolve to an IPv6 address.

##### Returns

On success, 0 is returned. On error, < 0 is returned.

## 7.5 Convert an IP Address Text String for an Interface

### 7.5.1 Function Documentation

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

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

##### Parameters

in	<i>hostname</i>	Pointer to an IP address string or host name string.
in	<i>addr</i>	Pointer to struct <a href="#">ip46addr</a> for the resolved IP address. The caller must specify which IP address (v4 or v6) it intends to resolve to: If <i>addr</i> type is AF_INET, resolve to an IPv4 address. If <i>addr</i> 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 is returned. On error, < 0 is returned.

## 7.6 Get a List of DNS Servers

### 7.6.1 Function Documentation

#### 7.6.1.1 `int32_t qapi_Net_DNSc_Get_Server_List ( qapi_Net_DNS_Server_List_t * svr_list, uint8_t iface_index )`

Gets the list of configured DNS servers.

##### 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 is returned. On error, -1 is returned.

## 7.7 Get Index for Added DNS Server

### 7.7.1 Function Documentation

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

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

##### 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 <i>svr_addr</i> is added.
in	<i>iface</i>	Pointer to the interface string on which the server is added.

##### Returns

On success, QAPI\_OK is returned. On error, -QAPI\_ERROR is returned.

## 7.8 Add a DNS Server

### 7.8.1 Function Documentation

#### 7.8.1.1 `int32_t qapi_Net_DNSc_Add_Server ( char * svr_addr, uint32_t id )`

Adds a DNS server to the system.

##### Parameters

in	<i>svr_addr</i>	Pointer to the DNS server's IP address string.
in	<i>id</i>	Server ID can be QAPI_NET_DNS_PRIMARY_SERVER_ID, QAPI_NET_DNS_SECONDARY_SERVER_ID, or QAPI_NET_DNS_ANY_SERVER_ID.

##### Returns

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



## 7.9 Add a DNS Server to an Interface

### 7.9.1 Function Documentation

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

Adds a DNS server to a PDN interface.

##### Parameters

in	<i>svr_addr</i>	Pointer to DNS server's IP address string.
in	<i>id</i>	Server ID can be QAPI_NET_DNS_PRIMARY_SERVER_ID, QAPI_NET_DNS_SECONDARY_SERVER_ID. or 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 is returned. On error, -1 is returned.

## 7.10 Remove a DNS Server

### 7.10.1 Function Documentation

#### 7.10.1.1 `int32_t qapi_Net_DNSc_Del_Server ( uint32_t id )`

Removes a DNS server from the system.

##### Parameters

in	<i>id</i>	Server ID can be QAPI_NET_DNS_PRIMARY_SERVER_ID or QAPI_NET_DNS_SECONDARY_SERVER_ID.
----	-----------	--

##### Returns

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

## 7.11 Removes a DNS Server from an Interface

### 7.11.1 Function Documentation

#### 7.11.1.1 `int32_t qapi_Net_DNSc_Del_Server_on_iface ( uint32_t id, char * iface_index )`

Removes a DNS server from an interface.

##### Parameters

in	<i>id</i>	Server ID can be QAPI_NET_DNS_PRIMARY_SERVER_ID or QAPI_NET_DNS_SECONDARY_SERVER_ID.
in	<i>iface_index</i>	Name of interface from which to delete a DNS server.

##### Returns

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

## 7.12 Get IPv4 Host Information by Name

### 7.12.1 Function Documentation

#### 7.12.1.1 `struct qapi_hostent_s* qapi_Net_DNSc_Get_Host_By_Name ( char * name )` [read]

Gets the 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.

##### Parameters

in	<i>name</i>	Pointer to either a host name or an IPv4 address in standard dot notation.
----	-------------	--

##### Returns

On success, a pointer to a hostent structure.

On error, NULL is returned.

## 7.13 Get IPv4/IPv6 Host Information by Name

### 7.13.1 Function Documentation

#### 7.13.1.1 `struct qapi_hostent_s* qapi_Net_DNSc_Get_Host_By_Name2 ( char * name, int32_t af ) [read]`

Gets the 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.

##### 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 <code>AF_INET</code> or <code>AF_INET6</code> .

##### Returns

On success, a pointer to a `hostent` structure.

On error, `NULL` is returned.

## 8 MQTT API

---

This chapter describes the MQTT API.

- [MQTT Data Types](#)
- [MQTT APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 8.1 MQTT Data Types

### Net MQTT Length Definitions

- #define [QAPI\\_NET\\_MQTT\\_MAX\\_CLIENT\\_ID\\_LEN](#) 23
- #define [QAPI\\_NET\\_MQTT\\_MAX\\_TOPIC\\_LEN](#) 128

### 8.1.1 Define Documentation

#### 8.1.1.1 #define QAPI\_NET\_MQTT\_MAX\_CLIENT\_ID\_LEN 23

Maximum client ID length.

#### 8.1.1.2 #define QAPI\_NET\_MQTT\_MAX\_TOPIC\_LEN 128

Maximum topic length.

### 8.1.2 Data Structure Documentation

#### 8.1.2.1 struct qapi\_Net\_MQTT\_config\_s

MQTT configuration structure.

##### Data fields

Type	Parameter	Description
struct <a href="#">sockaddr</a>	local	MQTT client IP address and port number.
struct <a href="#">sockaddr</a>	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 keep-alive duration in seconds.
uint8_t	clean_session	Clean session flag: 0 – no clean session; 1 – clean session.
uint8_t *	will_topic	Will topic name.
int32_t	will_topic_len	Will topic length.
uint8_t *	will_message	Will message.
int32_t	will_message_ len	Will message length
uint8_t	will_retained	Will retain flag.
uint8_t	will_qos	Will QOS.
uint8_t *	username	Client user name.
int32_t	username_len	Client user name length.
uint8_t *	password	Client password.
int32_t	password_len	Client password length.
qapi_Net_SSL- _Config_t	ssl_cfg	SSL configuration.

Type	Parameter	Description
<a href="#">qapi_Net_SSL- _CAList_t</a>	ca_list	SSL CA cert details.
<a href="#">qapi_Net_SSL- _Cert_t</a>	cert	SSL cert details.

### 8.1.3 Enumeration Type Documentation

#### 8.1.3.1 enum QAPI\_NET\_MQTT\_SUBSCRIBE\_CBK\_MSG

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.

#### 8.1.3.2 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.

#### 8.1.3.3 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.



## 8.2 MQTT APIs

### 8.2.1 Function Documentation

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

Creates a new MQTT context.

##### Parameters

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

##### Returns

QAPI\_OK on success, QAPI\_ERROR on failure.

#### 8.2.1.2 `qapi_Status_t qapi_Net_MQTT_Destroy ( qapi_Net_MQTT_Hndl_t hndl )`

Destroys an MQTT context.

##### Parameters

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

##### Returns

QAPI\_OK on success or QAPI\_ERROR on failure.

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

Connects to an MQTT broker.

##### Parameters

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

##### Returns

QAPI\_OK on success or < 0 on failure.

#### 8.2.1.4 `qapi_Status_t qapi_Net_MQTT_Disconnect ( qapi_Net_MQTT_Hndl_t hndl )`

Disconnects from an MQTT broker.

**Parameters**

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

**Returns**

QAPI\_OK on success or < 0 on failure.

**8.2.1.5** `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 )`

Publishes a message to a particular topic.

**Parameters**

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

**Returns**

QAPI\_OK on success or <0 on failure.

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

Subscribes to a topic.

**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.

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

Unsubscribes from a 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.

### 8.2.1.8 **qapi\_Status\_t qapi\_Net\_MQTT\_Set\_Connect\_Callback ( qapi\_Net\_MQTT\_Hndl\_t *hndl*, qapi\_Net\_MQTT\_Connect\_CB\_t *callback* )**

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

**Parameters**

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

**Returns**

Success or failure.

### 8.2.1.9 **qapi\_Status\_t qapi\_Net\_MQTT\_Set\_Subscribe\_Callback ( qapi\_Net\_MQTT\_Hndl\_t *hndl*, qapi\_Net\_MQTT\_Subscribe\_CB\_t *callback* )**

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

**Parameters**

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

**Returns**

QAPI\_OK on success or < 0 on failure.

### 8.2.1.10 **qapi\_Status\_t qapi\_Net\_MQTT\_Set\_Message\_Callback ( qapi\_Net\_MQTT\_Hndl\_t *hndl*, qapi\_Net\_MQTT\_Message\_CB\_t *callback* )**

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

**Parameters**

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

**Returns**

QAPI\_OK on success or  $< 0$  on failure.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 9 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.

- [HTTP\(S\) API](#)

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

## 9.1 HTTP(S) API

### 9.1.1 Data Structure Documentation

#### 9.1.1.1 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.

### 9.1.2 Typedef Documentation

#### 9.1.2.1 typedef void(\* qapi\_HTTPc\_CB\_t)(void \*arg, int32\_t state, void \*value)

HTTP response user callback registered during [qapi\\_Net\\_HTTPc\\_New\\_sess\(\)](#).

Parameters

in	arg	User payload information.
in	state	HTTP response state.
in	value	HTTP response information.

### 9.1.3 Enumeration Type Documentation

#### 9.1.3.1 enum qapi\_Net\_HTTPc\_Method\_e

HTTP request types supported by [qapi\\_Net\\_HTTPc\\_Request\(\)](#).

Enumerator:

**QAPI\_NET\_HTTP\_CLIENT\_GET\_E** HTTP get request.  
**QAPI\_NET\_HTTP\_CLIENT\_POST\_E** HTTP post request.  
**QAPI\_NET\_HTTP\_CLIENT\_PUT\_E** HTTP put request.  
**QAPI\_NET\_HTTP\_CLIENT\_PATCH\_E** HTTP patch request.  
**QAPI\_NET\_HTTP\_CLIENT\_HEAD\_E** HTTP head request.

#### 9.1.3.2 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 – server connection is closed.  
**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.

## 9.1.4 Function Documentation

### 9.1.4.1 `qapi_Status_t qapi_Net_HTTPc_Start ( void )`

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\(\)](#).

#### Returns

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

### 9.1.4.2 `qapi_Status_t qapi_Net_HTTPc_Stop ( void )`

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\(\)](#).

#### Returns

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

### 9.1.4.3 `qapi_Net_HTTPc_handle_t qapi_Net_HTTPc_New_sess ( uint32_t timeout, uint32_t ssl_ctx, qapi_HTTPc_CB_t callback, void * arg, uint16_t httpc_max_body_length, uint16_t httpc_max_header_length )`

Creates a new HTTP client session.

In order 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.

#### Parameters

in	<i>timeout</i>	Timeout (in ms) of a session method (zero is not recommended).
----	----------------	--

in	<i>ssl_ctx</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.

#### 9.1.4.4 **qapi\_Status\_t qapi\_Net\_HTTPc\_Free\_sess ( qapi\_Net\_HTTPc\_handle\_t handle )**

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.

### Parameters

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

### Returns

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

#### 9.1.4.5 **qapi\_Status\_t qapi\_Net\_HTTPc\_Connect ( qapi\_Net\_HTTPc\_handle\_t handle, const char \* URL, uint16\_t port )**

Connects an HTTP client session to the HTTP server.

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

### 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.



#### 9.1.4.6 **qapi\_Status\_t qapi\_Net\_HTTPc\_Disconnect ( qapi\_Net\_HTTPc\_handle\_t *handle* )**

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.

##### Parameters

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

##### Returns

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

#### 9.1.4.7 **qapi\_Status\_t qapi\_Net\_HTTPc\_Request ( qapi\_Net\_HTTPc\_handle\_t *handle*, qapi\_Net\_HTTPc\_Method\_e *cmd*, const char \* *URL* )**

Processes the HTTP client session requests and sends them to the HTTP server.

##### 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.

#### 9.1.4.8 **qapi\_Status\_t qapi\_Net\_HTTPc\_Set\_Body ( qapi\_Net\_HTTPc\_handle\_t *handle*, const char \* *body*, uint32\_t *body\_Length* )**

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.

##### 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.

#### 9.1.4.9 **qapi\_Status\_t qapi\_Net\_HTTPc\_Set\_Param ( qapi\_Net\_HTTPc\_handle\_t handle, const char \* key, const char \* value )**

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.

##### 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.

#### 9.1.4.10 **qapi\_Status\_t qapi\_Net\_HTTPc\_Add\_Header\_Field ( qapi\_Net\_HTTPc\_handle\_t handle, const char \* type, const char \* value )**

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.

##### 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.

#### 9.1.4.11 **qapi\_Status\_t qapi\_Net\_HTTPc\_Clear\_Header ( qapi\_Net\_HTTPc\_handle\_t handle )**

Clears an HTTP client session header.

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

##### Parameters

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

##### Returns

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

#### 9.1.4.12 **qapi\_Status\_t** qapi\_Net\_HTTPc\_Configure\_SSL ( **qapi\_Net\_HTTPc\_handle\_t** *handle*, **qapi\_Net\_SSL\_Config\_t** \* *ssl\_Cfg* )

Configures an HTTP client SSL session.

##### 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.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 10 QAPI Status and Error Codes

---

This chapter describes common and module-specific status and error codes.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 10.1 QAPI Status Codes

### SSL Module Error Codes

- #define [QAPI\\_ERR\\_SSL\\_CERT](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 1)
- #define [QAPI\\_ERR\\_SSL\\_CONN](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 2)
- #define [QAPI\\_ERR\\_SSL\\_HS\\_NOT\\_DONE](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 3)
- #define [QAPI\\_ERR\\_SSL\\_ALERT\\_RECV](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 4)
- #define [QAPI\\_ERR\\_SSL\\_ALERT\\_FATAL](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 5)
- #define [QAPI\\_SSL\\_HS\\_IN\\_PROGRESS](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 6)
- #define [QAPI\\_SSL\\_OK\\_HS](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 7)
- #define [QAPI\\_ERR\\_SSL\\_CERT\\_CN](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 8)
- #define [QAPI\\_ERR\\_SSL\\_CERT\\_TIME](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 9)
- #define [QAPI\\_ERR\\_SSL\\_CERT\\_NONE](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 10)
- #define [QAPI\\_ERR\\_SSL\\_NETBUF](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 11)
- #define [QAPI\\_ERR\\_SSL SOCK](#) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 12)

### Generic Error Codes

- #define [QAPI\\_NET\\_ERR\\_INVALID\\_IPADDR](#) ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 21)))
- #define [QAPI\\_NET\\_ERR\\_CANNOT\\_GET\\_SCOPEID](#) ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 22)))
- #define [QAPI\\_NET\\_ERR\\_SOCKET\\_CMD\\_TIME\\_OUT](#) ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 23)))
- #define [QAPI\\_NET\\_ERR\\_MAX\\_SERVER\\_REACHED](#) ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 24)))

### MQTT Error Codes

- #define [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) 25
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_ALLOC\\_FAILURE](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#)))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_BAD\\_PARAM](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 1))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_BAD\\_STATE](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_

- NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 2))
- #define QAPI\_NET\_MQTT\_ERR\_CONN\_CLOSED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 3))
  - #define QAPI\_NET\_MQTT\_ERR\_MSG\_DESERIALIZATION\_FAILURE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 4))
  - #define QAPI\_NET\_MQTT\_ERR\_MSG\_SERIALIZATION\_FAILURE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 5))
  - #define QAPI\_NET\_MQTT\_ERR\_NEGATIVE\_CONNACK ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 6))
  - #define QAPI\_NET\_MQTT\_ERR\_NO\_DATA ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 7))
  - #define QAPI\_NET\_MQTT\_ERR\_NONZERO\_REFCOUNT ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 8))
  - #define QAPI\_NET\_MQTT\_ERR\_PINGREQ\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 9))
  - #define QAPI\_NET\_MQTT\_ERR\_PUBACK\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 10))
  - #define QAPI\_NET\_MQTT\_ERR\_PUBCOMP\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 11))
  - #define QAPI\_NET\_MQTT\_ERR\_PUBLISH\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 12))
  - #define QAPI\_NET\_MQTT\_ERR\_PUBREC\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 13))
  - #define QAPI\_NET\_MQTT\_ERR\_PUBREL\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 14))
  - #define QAPI\_NET\_MQTT\_ERR\_RX\_INCOMPLETE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 15))
  - #define QAPI\_NET\_MQTT\_ERR\_SOCKET\_FATAL\_ERROR ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 16))

16))

- #define [QAPI\\_NET\\_MQTT\\_ERR\\_TCP\\_BIND\\_FAILED](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 17))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_TCP\\_CONNECT\\_FAILED](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 18))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_SSL\\_CONN\\_FAILURE](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 19))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_SUBSCRIBE\\_MSG\\_CREATION\\_FAILED](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 21))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_SUBSCRIBE\\_UNKNOWN\\_TOPIC](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 21))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_UNSUBSCRIBE\\_MSG\\_CREATION\\_FAILED](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 22))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_UNEXPECTED\\_MSG](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 23))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_PARTIAL\\_SUBSCRIPTION\\_FAILURE](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 24))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_RESTORE\\_FAILURE](#) ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, [QAPI\\_NET\\_MQTT\\_ERR\\_NUM\\_START](#) + 25))
- #define [QAPI\\_NET\\_MQTT\\_ERR\\_MAX\\_NUMS](#) 26

## QAPI Modules

The following definitions represent the IDs for the various modules of the QAPI.

Note that if OEMs wish to add their own module IDs, it is recommended to start at 100 to avoid possible conflicts with updates to the QAPI that adds in additional modules.

- #define [QAPI\\_MOD\\_BASE](#) (0)
- #define [QAPI\\_MOD\\_802\\_15\\_4](#) (1)
- #define [QAPI\\_MOD\\_NETWORKING](#) (2)
- #define [QAPI\\_MOD\\_WIFI](#) (3)
- #define [QAPI\\_MOD\\_BT](#) (4)
- #define [QAPI\\_MOD\\_BSP](#) (5)

- #define **QAPI\_MOD\_BSP\_I2C\_MASTER** (6)
- #define **QAPI\_MOD\_BSP\_SPI\_MASTER** (7)
- #define **QAPI\_MOD\_BSP\_TLMM** (8)
- #define **QAPI\_MOD\_BSP\_GPIPOINT** (9)
- #define **QAPI\_MOD\_BSP\_PWM** (10)
- #define **QAPI\_MOD\_BSP\_ERR** (11)
- #define **QAPI\_MOD\_BSP\_DIAG** (12)
- #define **QAPI\_MOD\_BSP\_OM\_SMEM** (13)
- #define **QAPI\_MOD\_CRYPT** (14)

### Common QAPI Status Codes

The following definitions represent the status codes common to all of the QAPI modules.

- #define **QAPI\_OK** ((qapi\_Status\_t)(0))
- #define **QAPI\_ERROR** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 1)))
- #define **QAPI\_ERR\_INVALID\_PARAM** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 2)))
- #define **QAPI\_ERR\_NO\_MEMORY** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 3)))
- #define **QAPI\_ERR\_NO\_RESOURCE** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 4)))
- #define **QAPI\_ERR\_BUSY** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 6)))
- #define **QAPI\_ERR\_NO\_ENTRY** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 7)))
- #define **QAPI\_ERR\_NOT\_SUPPORTED** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 8)))
- #define **QAPI\_ERR\_TIMEOUT** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 9)))
- #define **QAPI\_ERR\_BOUNDS** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 10)))
- #define **QAPI\_ERR\_BAD\_PAYLOAD** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 11)))
- #define **QAPI\_ERR\_EXISTS** ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 12)))

### 10.1.1 Define Documentation

#### 10.1.1.1 #define **QAPI\_ERR\_SSL\_CERT** \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 1)

Error in own certificate.



**10.1.1.2 #define QAPI\_ERR\_SSL\_CONN \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 2)**

Error with the SSL connection.

**10.1.1.3 #define QAPI\_ERR\_SSL\_HS\_NOT\_DONE \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 3)**

Handshake must be completed before the operation can be attempted.

**10.1.1.4 #define QAPI\_ERR\_SSL\_ALERT\_RECV \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 4)**

Received an SSL warning alert message.

**10.1.1.5 #define QAPI\_ERR\_SSL\_ALERT\_FATAL \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 5)**

Received an SSL fatal alert message.

**10.1.1.6 #define QAPI\_SSL\_HS\_IN\_PROGRESS \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 6)**

Handshake is in progress.

**10.1.1.7 #define QAPI\_SSL\_OK\_HS \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 7)**

Handshake was successful.

**10.1.1.8 #define QAPI\_ERR\_SSL\_CERT\_CN \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 8)**

The SSL certificate of the peer is trusted, CN matches the host name, time has expired.

**10.1.1.9 #define QAPI\_ERR\_SSL\_CERT\_TIME \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 9)**

The SSL certificate of the peer is trusted, CN does not match the host name, time is valid.

**10.1.1.10 #define QAPI\_ERR\_SSL\_CERT\_NONE \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 10)**

The SSL certificate of the peer is not trusted.

**10.1.1.11 #define QAPI\_ERR\_SSL\_NETBUF \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 11)**

Connection drops when out of network buffers; usually a resource configuration error.

**10.1.1.12    #define QAPI\_ERR\_SSL\_SOCKET \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 12)**

Socket error.

**10.1.1.13    #define QAPI\_NET\_ERR\_INVALID\_IPADDR ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 21)))**

IP address is invalid.

**10.1.1.14    #define QAPI\_NET\_ERR\_CANNOT\_GET\_SCOPEID ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 22)))**

Failed to get the scope ID.

**10.1.1.15    #define QAPI\_NET\_ERR\_SOCKET\_CMD\_TIME\_OUT ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 23)))**

Socket command timed out.

**10.1.1.16    #define QAPI\_NET\_ERR\_MAX\_SERVER\_REACHED ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, 24)))**

Maximum server address (v4/v6) reached in the server's list.

**10.1.1.17    #define QAPI\_NET\_MQTT\_ERR\_NUM\_START 25**

MQTT error number start.

**10.1.1.18    #define QAPI\_NET\_MQTT\_ERR\_ALLOC\_FAILURE ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START)**

Memory allocation failed.

**10.1.1.19    #define QAPI\_NET\_MQTT\_ERR\_BAD\_PARAM ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 1))**

Bad parameter while invoking the API.

**10.1.1.20    #define QAPI\_NET\_MQTT\_ERR\_BAD\_STATE ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 2))**

Connection is in a bad state.

**10.1.1.21** **#define QAPI\_NET\_MQTT\_ERR\_CONN\_CLOSED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 3))**

Connection is closed.

**10.1.1.22** **#define QAPI\_NET\_MQTT\_ERR\_MSG\_DESERIALIZATION\_FAILURE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 4))**

Packet decode failed.

**10.1.1.23** **#define QAPI\_NET\_MQTT\_ERR\_MSG\_SERIALIZATION\_FAILURE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 5))**

Packet encode failed.

**10.1.1.24** **#define QAPI\_NET\_MQTT\_ERR\_NEGATIVE\_CONNACK ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 6))**

Negative CONNACK received.

**10.1.1.25** **#define QAPI\_NET\_MQTT\_ERR\_NO\_DATA ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 7))**

No data.

**10.1.1.26** **#define QAPI\_NET\_MQTT\_ERR\_NONZERO\_REFCOUNT ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 8))**

No zero reference count while disconnecting.

**10.1.1.27** **#define QAPI\_NET\_MQTT\_ERR\_PINGREQ\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 9))**

Ping request message creation failed.

**10.1.1.28** **#define QAPI\_NET\_MQTT\_ERR\_PUBACK\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 10))**

PUBACK message creation failed.

**10.1.1.29** **#define QAPI\_NET\_MQTT\_ERR\_PUBCOMP\_MSG\_CREATION\_FAILED** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 11))

PUBCOMP message creation failed.

**10.1.1.30** **#define QAPI\_NET\_MQTT\_ERR\_PUBLISH\_MSG\_CREATION\_FAILED** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 12))

Publish message creation failed.

**10.1.1.31** **#define QAPI\_NET\_MQTT\_ERR\_PUBREC\_MSG\_CREATION\_FAILED** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 13))

PUBREC message creation failed.

**10.1.1.32** **#define QAPI\_NET\_MQTT\_ERR\_PUBREL\_MSG\_CREATION\_FAILED** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 14))

PUBREL message creation failed.

**10.1.1.33** **#define QAPI\_NET\_MQTT\_ERR\_RX\_INCOMPLETE** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 15))

Rx is incomplete.

**10.1.1.34** **#define QAPI\_NET\_MQTT\_ERR\_SOCKET\_FATAL\_ERROR** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 16))

Socket fatal error.

**10.1.1.35** **#define QAPI\_NET\_MQTT\_ERR\_TCP\_BIND\_FAILED** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 17))

TCP bind error.

**10.1.1.36** **#define QAPI\_NET\_MQTT\_ERR\_TCP\_CONNECT\_FAILED** ((qapi\_Status\_t) \_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 18))

TCP connection error.

**10.1.1.37** **#define QAPI\_NET\_MQTT\_ERR\_SSL\_CONN\_FAILURE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 19))**

SSL connection failed.

**10.1.1.38** **#define QAPI\_NET\_MQTT\_ERR\_SUBSCRIBE\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 21))**

Subscribe message creation failed.

**10.1.1.39** **#define QAPI\_NET\_MQTT\_ERR\_SUBSCRIBE\_UNKNOWN\_TOPIC ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 21))**

Subscribe unknown topic.

**10.1.1.40** **#define QAPI\_NET\_MQTT\_ERR\_UNSUBSCRIBE\_MSG\_CREATION\_FAILED ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 22))**

Unsubscribe message creation failed.

**10.1.1.41** **#define QAPI\_NET\_MQTT\_ERR\_UNEXPECTED\_MSG ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 23))**

Unexpected message was received.

**10.1.1.42** **#define QAPI\_NET\_MQTT\_ERR\_PARTIAL\_SUBSCRIPTION\_FAILURE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 24))**

Subscription failed.

**10.1.1.43** **#define QAPI\_NET\_MQTT\_ERR\_RESTORE\_FAILURE ((qapi\_Status\_t)\_\_QAPI\_ERROR(QAPI\_MOD\_NETWORKING, QAPI\_NET\_MQTT\_ERR\_NUM\_START + 25))**

Restore failed.

**10.1.1.44** **#define QAPI\_NET\_MQTT\_ERR\_MAX\_NUMS 26**

Maximum error number.

**10.1.1.45 #define QAPI\_OK ((qapi\_Status\_t)(0))**

Success.

**10.1.1.46 #define QAPI\_ERROR ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 1)))**

General error.

**10.1.1.47 #define QAPI\_ERR\_INVALID\_PARAM ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 2)))**

Invalid parameter.

**10.1.1.48 #define QAPI\_ERR\_NO\_MEMORY ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 3)))**

Memory allocation error.

**10.1.1.49 #define QAPI\_ERR\_NO\_RESOURCE ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 4)))**

Resource allocation error.

**10.1.1.50 #define QAPI\_ERR\_BUSY ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 6)))**

Operation is busy.

**10.1.1.51 #define QAPI\_ERR\_NO\_ENTRY ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 7)))**

Entry was not found.

**10.1.1.52 #define QAPI\_ERR\_NOT\_SUPPORTED ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 8)))**

Feature is not supported.

**10.1.1.53 #define QAPI\_ERR\_TIMEOUT ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 9)))**

Operation timed out.

**10.1.1.54 #define QAPI\_ERR\_BOUNDS ((qapi\_Status\_t)(\_\_QAPI\_ERROR(QAPI\_MOD\_BASE, 10)))**

Out of bounds.

**10.1.1.55** `#define QAPI_ERR_BAD_PAYLOAD ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 11)))`

Bad payload.

**10.1.1.56** `#define QAPI_ERR_EXISTS ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 12)))`

Entry already exists.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

# 11 System Drivers APIs

---

This chapter describes the GPIO interrupt controller and the pin mode multiplexer (PMM) APIs.

- [GPIO Interrupt Controller APIs](#)
- [PMM APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com



## 11.1 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.
```

```
qapi_Status_t      nStatus;
qapi_Instance_Handle_t pH;
uint32_t           nLoopCounter = 0;

nStatus = qapi_GPIOINT_Register_Interrupt(&pH,           // Pass in a pointer
                                          // to the handle
                                          10,             // GPIO 10 is used
                                          pfnCallback,     // Callback fn pointer
                                          NULL,            // NULL callback data
                                          GPIOINT_TRIGGER_HIGH_LEVEL,
                                                          // Level high trigger
                                          QAPI_GPIOINT_PRIO_MEDIUM_E,
                                                          // Priority of interrupt
                                          false );         // Maskable Interrupt

if ( nStatus != QAPI_OK )
{
    // Error!
}

// Trigger interrupt for GPIO 10
nStatus = qapi_GPIOINT_Trigger_Interrupt( &pH, 10 );
if ( nStatus != QAPI_OK )
{
    // Error!
}

while ( nLoopCounter++ < 1000 )
{
}
```

```
// Deregister the GPIO Interrupt
nRet = qapi_GPIoint_Deregister_Interrupt( &pH, 10 );
if ( nRet != GPIoint_SUCCESS )
{
    // Error!
}
```

## 11.1.1 Typedef Documentation

### 11.1.1.1 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.

### 11.1.1.2 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.

### 11.1.1.3 typedef void\* qapi\_Instance\_Handle\_t

GPIO interrupt handle definition.

## 11.1.2 Enumeration Type Documentation

### 11.1.2.1 enum qapi\_GPIoint\_Trigger\_e

GPIO interrupt trigger type enumeration for supported triggers.

Enumerator:

**QAPI\_GPIoint\_TRIGGER\_LEVEL\_HIGH\_E** Level triggered active high.  
**QAPI\_GPIoint\_TRIGGER\_LEVEL\_LOW\_E** Level triggered active low.  
**QAPI\_GPIoint\_TRIGGER\_EDGE\_RISING\_E** Rising edge triggered.  
**QAPI\_GPIoint\_TRIGGER\_EDGE\_FALLING\_E** Falling edge triggered.  
**QAPI\_GPIoint\_TRIGGER\_EDGE\_DUAL\_E** Dual edge triggered.

### 11.1.2.2 enum qapi\_GPIOINT\_Priority\_e

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

Enumerator:

**QAPI\_GPIOINT\_PRIO\_HIGHEST\_E** Highest priority.  
**QAPI\_GPIOINT\_PRIO\_HIGH\_E** Medium-high priority.  
**QAPI\_GPIOINT\_PRIO\_MEDIUM\_E** Medium priority.  
**QAPI\_GPIOINT\_PRIO\_LOW\_E** Medium-low priority.  
**QAPI\_GPIOINT\_PRIO\_LOWEST\_E** Highest priority.

### 11.1.3 Function Documentation

#### 11.1.3.1 qapi\_Status\_t qapi\_GPIOINT\_Register\_Interrupt ( qapi\_Instance\_Handle\_t \* *pH*, uint32\_t *nGpio*, qapi\_GPIOINT\_CB\_t *pfnCallback*, qapi\_GPIOINT\_Callback\_Data\_t *nData*, qapi\_GPIOINT\_Trigger\_e *eTrigger*, qapi\_GPIOINT\_Priority\_e *ePriority*, qbool\_t *bNmi* )

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.

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.

### 11.1.3.2 **qapi\_Status\_t qapi\_GPIOINT\_Deregister\_Interrupt ( qapi\_Instance\_Handle\_t \* *pH*, uint32\_t *nGpio* )**

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.

#### Parameters

in	<i>pH</i>	Input handle to the client context.
in	<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.

### 11.1.3.3 **qapi\_Status\_t qapi\_GPIOINT\_Set\_Trigger ( qapi\_Instance\_Handle\_t \* *pH*, uint32\_t *nGpio*, qapi\_GPIOINT\_Trigger\_e *eTrigger* )**

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.

#### 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.

### 11.1.3.4 **qapi\_Status\_t qapi\_GPIOINT\_Enable\_Interrupt ( qapi\_Instance\_Handle\_t \* *pH*, uint32\_t *nGpio* )**

Enables a currently disabled and registered GPIO interrupt.

This is used primarily to unmask interrupts.

**Parameters**

in	<i>pH</i>	Input handle to the client context.
in	<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.

### 11.1.3.5 **qapi\_Status\_t qapi\_GPIOINT\_Disable\_Interrupt ( qapi\_Instance\_Handle\_t \* *pH*, uint32\_t *nGpio* )**

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.

**Parameters**

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

### 11.1.3.6 **qapi\_Status\_t qapi\_GPIOINT\_Trigger\_Interrupt ( qapi\_Instance\_Handle\_t \* *pH*, uint32\_t *nGpio* )**

Manually triggers a GPIO interrupt by writing to the appropriate register.

**Parameters**

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

### 11.1.3.7 qapi\_Status\_t qapi\_GPIOINT\_Is\_Interrupt\_Pending ( qapi\_Instance\_Handle\_t \* *pH*, uint32\_t *nGpio*, qbool\_t \* *pbIsPending* )

Queries to see if an interrupt is pending in the hardware by reading the appropriate register.

#### Parameters

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

#### Returns

- QAPI\_ERR\_INVALID\_PARAM – *pbIsPending* 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.

## 11.2 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.

qapi_GPIO_ID_t      gpio_id;
qapi_TLMM_Config_t  tlmm_config;
qapi_Status_t       status = QAPI_OK;

tlmm_config.pin = 13;
tlmm_config.func = 1           // Using the functionality tied to
                               // pin mux value 1

tlmm_config.dir = QAPI_GPIO_INPUT_E;
tlmm_config.pull = QAPI_GPIO_PULL_DOWN_E;
tlmm_config.drive = QAPI_GPIO_2MA_E; // drive is for output pins, specify
                                     // the default here

status = qapi_TLMM_Get_Gpio_ID( &tlmm_config, &gpio_id);

if (status == QAPI_OK)
{
    status = qapi_TLMM_Config_Gpio(gpio_id, &tlmm_config);

    if (status != QAPI_OK)
    {
        // Handle failed case here
    }
}
```

## 11.2.1 Data Structure Documentation

### 11.2.1.1 struct qapi\_TLMM\_Config\_t

GPIO configuration.

This structure is used to specify the configuration for a GPIO on the SoC. The GPIO can be configured as an 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.

## 11.2.2 Typedef Documentation

### 11.2.2.1 typedef uint16\_t qapi\_GPIO\_ID\_t

SoC pin 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.

## 11.2.3 Enumeration Type Documentation

### 11.2.3.1 enum qapi\_GPIO\_Direction\_t

Pin direction enumeration.

The enumeration is used to specify 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.

### 11.2.3.2 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.

**11.2.3.3 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.

**11.2.3.4 enum qapi\_GPIO\_Value\_t**

GPIO output state specification.

This enumeration specifies the value to write to a GPIO pin configured as an 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.

**11.2.4 Function Documentation****11.2.4.1 qapi\_Status\_t qapi\_TLMM\_Get\_Gpio\_ID ( qapi\_TLMM\_Config\_t \*  
qapi\_TLMM\_Config, qapi\_GPIO\_ID\_t \* qapi\_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.

**Parameters**

in	qapi_TLMM_Config	Pointer to the pin configuration data.
in	qapi_GPIO_ID	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.

#### 11.2.4.2 **qapi\_Status\_t qapi\_TLMM\_Release\_Gpio\_ID ( qapi\_TLMM\_Config\_t \* qapi\_TLMM\_Config, qapi\_GPIO\_ID\_t qapi\_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.

**Parameters**

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

**Returns**

- QAPI\_OK – Pin was released successfully.
- QAPI\_ERR – Pin could not be released.

#### 11.2.4.3 **qapi\_Status\_t qapi\_TLMM\_Config\_Gpio ( qapi\_GPIO\_ID\_t qapi\_GPIO\_ID, qapi\_TLMM\_Config\_t \* qapi\_TLMM\_Config )**

Changes the SoC pin configuration.

This function configures an SoC pin based on a set of fields specified in the configuration structure reference passed in as a parameter.

**Parameters**

in	<i>qapi_GPIO_ID</i>	Pin access ID retrieved from the <a href="#">qapi_TLMM_Get_Gpio_ID()</a> 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.

#### 11.2.4.4 **qapi\_Status\_t qapi\_TLMM\_Drive\_Gpio ( qapi\_GPIO\_ID\_t *qapi\_GPIO\_ID*, uint32\_t *pin*, qapi\_GPIO\_Value\_t *value* )**

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.

##### Parameters

in	<i>qapi_GPIO_ID</i>	Pin access ID retrieved from the <a href="#">qapi_TLMM_Get_Gpio_ID()</a> 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.

#### 11.2.4.5 **qapi\_Status\_t qapi\_TLMM\_Read\_Gpio ( qapi\_GPIO\_ID\_t *qapi\_GPIO\_ID*, uint32\_t *pin*, qapi\_GPIO\_Value\_t \* *qapi\_GPIO\_Value* )**

Reads the state of an SoC pin configured as an input GPIO.

##### Parameters

in	<i>qapi_GPIO_ID</i>	Pin access ID retrieved from the <a href="#">qapi_TLMM_Get_Gpio_ID()</a> call.
in	<i>pin</i>	SoC pin number to configure.
out	<i>qapi_GPIO_Value</i>	GIPO pin value.

##### Returns

- QAPI\_GPIO\_HIGH\_VALUE – Read value was HIGH.
- QAPI\_GPIO\_LOW\_VALUE – Read value was LOW.

## 12 Diagnostic Services Module

---

This chapter describes the diagnostic (Diag) services APIs.

- [QAPI Diag Services APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 12.1 QAPI Diag Services APIs

### 12.1.1 Define Documentation

#### 12.1.1.1 **#define QAPI\_DIAGPKT\_DISPATCH\_TABLE\_REGISTER( *xx\_subsysid*, *xx\_entry*, *inbuf*, *inbuf\_len*, *outbuf*, *outbuf\_len* )**

Macro to register the user space client's dispatch table with the diagnostics packet dispatching service.

The client must maintain two buffers (inbuf and outbuf) and must pass a pointer to these buffers while registering its user table with Diag. When the command is received from the tool for the user space client, Diag will copy the command to the inbuf of the client and call its handler with the length of the command that was written. The client must copy the response of the command to its outbuf and return the length of the response that was written or commit the response using `qapi_diagpkt_commit` with `IMMEDIATE_RSP_FLAG` and return 0.

**Note:** When a client command handler is processing a response, if `qapi_diagpkt_commit` is used, it returns only 0. For any other valid return length, Diag generates a response other than the one that is already committed.

Parameters:

- `xx_subsysid` – Subsystem ID of the client.
- `xx_entry` – Client registration table of type `diagpkt_user_table_entry_type` with the `func_ptr` field as `NULL` and `user_func_ptr` with the command handler.
- `inbuf` – Client static buffer to which Diag copies the command.
- `inbuf_len` – Client input static buffer length.
- `outbuf` – Client static buffer to which the client is to copy the response to the command.
- `outbuf_len` – Client output static buffer length.

Returns QAPI status; see [QAPI Status Codes](#).

#### 12.1.1.2 **#define QAPI\_DIAGPKT\_DISPATCH\_TABLE\_REGISTER\_V2\_DELAY( *xx\_cmdcode*, *xx\_subsysid*, *xx\_entry*, *inbuf*, *inbuf\_len*, *outbuf*, *outbuf\_len* )**

Macro to register the user space client's dispatch table of the delayed responses type with the diagnostics packet dispatching service.

The client must maintain two buffers (inbuf and outbuf) and must pass the pointers to these buffers while registering its user table with Diag.

When the command is received from the tool for the user space client, Diag copies the command to inbuf of the client and call its handler with the length of the command written. The client must copy the response to its outbuf and commit the immediate response using `qapi_diagpkt_commit` with `IMMEDIATE_RSP_FLAG`. Subsequent delayed responses must be committed using `qapi_diagpkt_commit` with `DELAYED_RSP_FLAG`.

**Note:** When a client command handler is processing a response, if `qapi_diagpkt_commit` is used, it returns only 0. For any other valid return length, Diag generates a response other than the one that is already committed.

Parameters:

- `xx_cmdcode` – Set to `DIAG_SUBSYS_CMD_VER_2_F` to specify that the table is being registered for delayed response functionality.
- `xx_subsysid` – Subsystem ID of the client.
- `xx_entry` – Client registration table of type `diagpkt_user_table_entry_type` with the `func_ptr` field as `NULL` and `user_func_ptr` with the command handler.
- `inbuf` – Client static buffer to which Diag copies the command.
- `inbuf_len` – Client input static buffer length.
- `outbuf` – Client static buffer to which the client is to copy the response to the command.
- `outbuf_len` – Client output static buffer length.

Returns QAPI status; see [QAPI Status Codes](#).

### 12.1.1.3 **#define QAPI\_MSG( xx\_ss\_id, xx\_ss\_mask, xx\_fmt, ... )**

Macro to log a client's `printf_type` messages with 0 to 9 parameters.

Parameters:

- `xx_ss_id` – Subsystem ID of the client.
- `xx_ss_mask` – Subsystem mask for this message (represents the logging level).
- `xx_fmt` – Format string.
- `xx_args` – Integer arguments.

Returns QAPI status; see [QAPI Status Codes](#).

### 12.1.1.4 **#define QAPI\_MSG\_SPRINTF( xx\_ss\_id, xx\_ss\_mask, xx\_fmt, ... )**

Macro to log a client's `sprintf_type` messages with 0 to 9 parameters.

Parameters:

- `xx_ss_id` – Subsystem ID of the client.
- `xx_ss_mask` – Subsystem mask for this message (represents the logging level).
- `xx_fmt` – Format string.
- `xx_args` – Arguments (integer or string type).

Returns QAPI status; see [QAPI Status Codes](#).

## 12.1.2 Data Structure Documentation

### 12.1.2.1 **struct qapi\_diagpkt\_subsys\_hdr\_type\_t**

Diag subsystem header structure for subsystem commands and responses.

**Data fields**

Type	Parameter	Description
uint8	command_code	Command code.

Type	Parameter	Description
uint8	subsys_id	Subsystem ID.
uint16	subsys_cmd_code	Subsystem command code.

### 12.1.2.2 struct qapi\_diagpkt\_subsys\_hdr\_type\_v2\_t

Diag subsystem v2 header structure for delayed subsystem responses.

#### Data fields

Type	Parameter	Description
uint8	command_code	Command code.
uint8	subsys_id	Subsystem ID.
uint16	subsys_cmd_code	Subsystem command code.
uint32	status	Can be used by clients for any error codes.
uint16	delayed_rsp_id	Unique ID for a set of delayed responses generated by a single command.
uint16	rsp_cnt	Increment this count for every response to track the sequence of responses.

## 12.1.3 Function Documentation

### 12.1.3.1 qapi\_Status\_t qapi\_user\_space\_tbl\_reg\_append\_proc ( diagpkt\_master\_table\_type \* tbl\_ptr, diagpkt\_user\_space\_table\_type \* user\_space\_tbl\_ptr )

Registers the user table given to the diagpkt master table and creates a new entry in diagpkt\_user\_space\_table with user\_space\_tbl\_ptr. Updates the port field of the master table entry with the index of its entry in diagpkt\_user\_space\_table.

#### Parameters

in	tbl_ptr	Structure for the diagnostics packet service master table to hold the client's table entries when the clients registers with the diagnostics packet services.
in	user_space_tbl_ptr	Structure for the diagnostics packet service user space table to hold the client's buffer details when the client registers with the diagnostics packet services.

#### Returns

QAPI status; see [QAPI Status Codes](#).

### 12.1.3.2 **qapi\_Status\_t qapi\_diagpkt\_get\_next\_delayed\_rsp\_id ( uint16\_t \* *delayed\_rsp\_id* )**

Gets a unique delayed response ID that is to be used for the set of delayed responses generated for a single command.

#### Parameters

in	<i>delayed_rsp_id</i>	Address of the variable that will be updated with the delayed response ID from Diag.
----	-----------------------	--

#### Returns

QAPI status; see [QAPI Status Codes](#).

### 12.1.3.3 **qapi\_Status\_t qapi\_diagpkt\_commit ( uint32\_t \* *outbuf*, uint32\_t *rsp\_len*, uint32\_t *rsp\_flag* )**

Processes the user space client's response and commits the response if all the sanity checks are passed. In the case of a failure, it generates an error response.

#### Parameters

in	<i>outbuf</i>	Client static buffer to which the client is to copy the response to the command.
in	<i>rsp_len</i>	Length of the response copied to outbuf.
in	<i>rsp_flag</i>	Flag that represents the type of response (immediate or delayed) or any error code.

#### Returns

QAPI status; see [QAPI Status Codes](#).

### 12.1.3.4 **qapi\_Status\_t qapi\_msg\_send ( const msg\_const\_type \* *const\_blk*, uint32\_t *num\_args*, ... )**

Internal API that is not to be used by clients directly. Use the [QAPI\\_MSG\(\)](#) macro to log a debug message. There are also arguments under a *va\_args* parameter (integer type) that are not shown in the protocol.

#### Parameters

in	<i>const_blk</i>	Constant information stored for a message.
in	<i>num_args</i>	Number of arguments for the message.

#### Returns

QAPI status; see [QAPI Status Codes](#).



### 12.1.3.5 `qapi_Status_t qapi_msg_sprintf ( const msg_const_type * const_blk, uint32_t num_args, ... )`

Internal API is not to be used by clients directly. Use the `QAPI_MSG_SPRINTD()` macro to log a debug message. There are also arguments under a `va_args` parameter (integer or string type) that are not shown in the protocol.

#### Parameters

in	<i>const_blk</i>	Constant information stored for a message.
in	<i>num_args</i>	Number of arguments for the message.

#### Returns

QAPI status; see [QAPI Status Codes](#).

### 12.1.3.6 `qapi_Status_t qapi_log_submit ( void * ptr )`

Logs an accumulated log entry. Header contents must be assigned by the caller before calling this function. Therefore, [qapi\\_log\\_set\\_code\(\)](#), [qapi\\_log\\_set\\_length\(\)](#), and [qapi\\_log\\_set\\_timestamp\(\)](#) must be used before this call.

#### Parameters

in	<i>ptr</i>	Pointer to the client-allocated log packet.
----	------------	---

#### Returns

QAPI status; see [QAPI Status Codes](#).

### 12.1.3.7 `qapi_Status_t qapi_log_set_length ( void * ptr, log_code_type length )`

Sets the length field in the specified log record.

#### Parameters

in	<i>ptr</i>	Pointer to the client-allocated log packet.
in	<i>length</i>	Length of the client-allocated log packet.

#### Returns

QAPI status; see [QAPI Status Codes](#).

### 12.1.3.8 `qapi_Status_t qapi_log_set_code ( void * ptr, log_code_type code )`

Sets the code field in the specified log record.

**Parameters**

in	<i>ptr</i>	Pointer to the client-allocated log packet.
in	<i>code</i>	Log code of the client-allocated log packet.

**Returns**

QAPI status; see [QAPI Status Codes](#).

**12.1.3.9 qapi\_Status\_t qapi\_log\_set\_timestamp ( void \* *plog\_hdr\_ptr* )**

Sets the timestamp field in the specified log record.

**Parameters**

in	<i>plog_hdr_ptr</i>	Pointer to the client-allocated log packet.
----	---------------------	---

**Returns**

QAPI status; see [QAPI Status Codes](#).

**12.1.3.10 qapi\_Status\_t qapi\_log\_status ( log\_code\_type *code* )**

Checks whether a particular code is enabled for logging.

**Parameters**

in	<i>code</i>	Log code of the client-allocated log packet.
----	-------------	--

**Returns**

QAPI status; see [QAPI Status Codes](#).

**12.1.3.11 qapi\_Status\_t qapi\_event\_report ( event\_id\_enum\_type *event\_id* )**

Reports an event without a payload.

**Parameters**

in	<i>event_id</i>	Event ID of the event to be reported.
----	-----------------	---------------------------------------

**Returns**

QAPI status; see [QAPI Status Codes](#).

### 12.1.3.12 **qapi\_Status\_t** qapi\_event\_report\_payload ( **event\_id\_enum\_type** *event\_id*, **uint8** *length*, **void \*** *data* )

Reports an event with a payload.

#### Parameters

in	<i>event_id</i>	Event ID of the event to be reported.
in	<i>length</i>	Length of the event to be reported.
in	<i>data</i>	Payload of the event to be reported.

#### Returns

QAPI status; see [QAPI Status Codes](#).

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

# 13 Storage Module

---

This chapter describes the file system data types and APIs.

- [File System Data Types](#)
- [File System APIs](#)
- [FTL Data Types and APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 13.1 File System Data Types

### 13.1.1 Data Structure Documentation

#### 13.1.1.1 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.

#### 13.1.1.2 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.

Type	Parameter	Description
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_DESCRIPTOR.
enum <a href="#">qapi_F-S_Filename_-Rule_e</a>	f_name_rule	File naming rule.
enum <a href="#">qapi_F-S_Filename_-Encoding_e</a>	f_name_-encoding	Encoding scheme.

### 13.1.1.3 struct qapi\_FS\_Iter\_Entry\_s

See the [qapi\\_FS\\_Iter\\_Next\(\)](#) API for information about this structure.

#### Data fields

Type	Parameter	Description
char	file_Path	Name of the directory component.
struct <a href="#">qapi_FS-Stat_Type_s</a>	SBuf	See <a href="#">qapi_FS_Stat_Type_s</a> for information on this structure.
uint32	qapi_FS_D_-Stats_Present	Bitmask for the <a href="#">qapi_FS_Stat_Type_s</a> structure that defines which fields are filled when the <a href="#">qapi_FS_Iter_Next()</a> API is called.

## 13.1.2 Enumeration Type Documentation

### 13.1.2.1 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.

### 13.1.2.2 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.

## 13.2 File System APIs

### 13.2.1 Function Documentation

#### 13.2.1.1 `qapi_FS_Status_t qapi_FS_Open_With_Mode ( const char * Path, int Oflag, qapi_FS_Mode_t Mode, int * Fd_ptr )`

Opens a file as per the specified Oflag and mode.

##### Parameters

in	<i>Path</i>	Path of the file that is to be opened.
in	<i>Oflag</i>	Argument that describes how this file is to be opened. It contains one of the following values: <ul style="list-style-type: none"> <li>QAPI_FS_O_RDONLY_E – Open for read only.</li> <li>QAPI_FS_O_WRONLY_E – Open for write only.</li> <li>QAPI_FS_O_RDWR_E – Open for read and write. In addition, the following flags can be bitwise ORed with this value: <ul style="list-style-type: none"> <li>QAPI_FS_O_APPEND_E – All writes will self-seek to the end of the file before writing.</li> <li>QAPI_FS_O_CREAT_E – Create the file if it does not exist.</li> <li>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: <ul style="list-style-type: none"> <li>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.</li> <li>QAPI_FS_O_CREAT_E   QAPI_FS_O_EXCL_E – Creates a file but fails if it already exists.</li> </ul> </li> </ul> </li> </ul>
in	<i>Mode</i>	If QAPI_FS_O_CREAT is a part of Oflag, a third argument (Mode) must be passed to <a href="#">qapi_FS_Open()</a> to define the file permissions given to the newly created file. If QAPI_FS_O_CREAT is not a part of flag, set Mode=0. One or more of the following permission bits can be ORed as the Mode parameter: <ul style="list-style-type: none"> <li>QAPI_FS_S_IRUSR_E – Read permission for a user</li> <li>QAPI_FS_S_IWUSR_E – Write permission for a user</li> <li>QAPI_FS_S_IXUSR_E – Execute permission for a user</li> <li>QAPI_FS_S_IRGRP_E – Read permission for a group</li> <li>QAPI_FS_S_IWGRP_E – Write permission for a group</li> <li>QAPI_FS_S_IXGRP_E – Execute permission for a group</li> <li>QAPI_FS_S_IROTH_E – Read permission for other</li> <li>QAPI_FS_S_IWOTH_E – Write permission for other</li> <li>QAPI_FS_S_IXOTH_E – Execute permission for other</li> <li>QAPI_FS_S_ISUID_E – Set UID on execution</li> <li>QAPI_FS_S_ISGID_E – Set GID on execution</li> <li>QAPI_FS_S_ISVTX_E – Sticky bit (hidden attribute in FAT)</li> </ul>



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

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.

### 13.2.1.2 qapi\_FS\_Status\_t qapi\_FS\_Open ( const char \* *Path*, int *Oflag*, int \* *Fd\_ptr* )

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.

## 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 <a href="#">qapi_FS_Open_With_Mode()</a> .
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\(\)](#).

### 13.2.1.3 **qapi\_FS\_Status\_t qapi\_FS\_Read ( int *Fd*, uint8 \* *Buf*, uint32 *Count*, uint32 \* *Bytes\_Read\_Ptr* )**

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.

**Parameters**

in	<i>Fd</i>	File descriptor obtained via the <a href="#">qapi_FS_Open()</a> 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.

### 13.2.1.4 **qapi\_FS\_Status\_t qapi\_FS\_Write ( int *Fd*, const uint8 \* *Buf*, uint32 *Count*, uint32 \* *Bytes\_Written\_Ptr* )**

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.

**Parameters**

in	<i>Fd</i>	File descriptor obtained via the <a href="#">qapi_FS_Open()</a> 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.

### 13.2.1.5 qapi\_FS\_Status\_t qapi\_FS\_Close ( int *Fd* )

Closes the file descriptor.

The descriptor will no longer refer to any file and will be allocated to subsequent calls to [qapi\\_FS\\_Open\(\)](#).

#### Parameters

in	<i>Fd</i>	File descriptor obtained via the <a href="#">qapi_FS_Open()</a> function.
----	-----------	---

#### Returns

Returns QAPI\_OK on success and -ve erro code is returned on failure.

### 13.2.1.6 qapi\_FS\_Status\_t qapi\_FS\_Rename ( const char \* *Old\_Path*, const char \* *New\_Path* )

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).

#### Parameters

in	<i>Old_Path</i>	Path name before the rename operation.
in	<i>New_Path</i>	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 accross different file systems is not permitted.
- QAPI\_ERR\_ENOTEMPTY – The *Old\_Path* directory is not empty.

### 13.2.1.7 qapi\_FS\_Status\_t qapi\_FS\_Truncate ( const char \* *Path*, qapi\_FS\_Offset\_t *Length* )

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.

**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.

### 13.2.1.8 **qapi\_FS\_Status\_t qapi\_FS\_Seek ( int *Fd*, qapi\_FS\_Offset\_t *Offset*, int *Whence*, qapi\_FS\_Offset\_t \* *Actual\_Offset\_Ptr* )**

Changes the file offset for the opened file descriptor.

Changing the file offset does not modify the file. If you lseek past the end of the file and then write, the gap will be filled with zero bytes. This gap may not actually allocate space. Using this API file can be seeked up to (4 GB -1) offset.

**Parameters**

in	<i>Fd</i>	File descriptor obtained via the <a href="#">qapi_FS_Open()</a> 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.

### 13.2.1.9 **qapi\_FS\_Status\_t qapi\_FS\_Mk\_Dir ( const char \* *Path*, qapi\_FS\_Mode\_t *Mode* )**

Creates a new directory.

#### Parameters

in	<i>Path</i>	Path for the directory.
in	<i>Mode</i>	Permission bits of the new directory. See the <a href="#">qapi_FS_Open()</a> API for information on Mode bits.

#### Returns

Returns QAPI\_OK on success and -ve error code is returned on failure.

- QAPI\_ERR\_ENOENT – No such 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.

### 13.2.1.10 **qapi\_FS\_Status\_t qapi\_FS\_Rm\_Dir ( const char \* *Path* )**

Removes a directory. Only empty directories can be removed.

#### 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.

### 13.2.1.11 **qapi\_FS\_Status\_t qapi\_FS\_Unlink ( const char \* *Path* )**

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.

**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 such path was found.
- QAPI\_ERR\_EPERM – Permission is denied.
- QAPI\_ERR\_ETXTBSY – The file is in use or open.
- QAPI\_ERR\_EINVAL – Invalid parameter.

### 13.2.1.12 **qapi\_FS\_Status\_t qapi\_FS\_Stat ( const char \* *Path*, struct qapi\_FS\_Stat\_Type\_s \* *SBuf* )**

Returns the statistics of a file.

**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 <a href="#">qapi_FS_Stat_Type_s</a> .

**Returns**

Returns QAPI\_OK on success and -ve error code is returned on failure.

### 13.2.1.13 **qapi\_FS\_Status\_t qapi\_FS\_Stat\_With\_Handle ( int *Fd*, struct qapi\_FS\_Stat\_Type\_s \* *SBuf* )**

Obtains information about the file with its open file handle.

**Parameters**

in	<i>Fd</i>	Handle to a file obtained using the <a href="#">qapi_FS_Open()</a> API.
out	<i>SBuf</i>	Information is returned in the structure <a href="#">qapi_FS_Stat_Type_s</a> .

**Returns**

Returns QAPI\_OK on success and -ve error code is returned on failure.

### 13.2.1.14 **qapi\_FS\_Status\_t qapi\_FS\_Statvfs ( const char \* *Path*, struct qapi\_FS\_Statvfs\_Type\_s \* *SBuf* )**

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](#).

#### 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 <a href="#">qapi_FS_Statvfs_Type_s</a> .

#### Returns

Returns QAPI\_OK on success, and -ve error code is returned on failure.

### 13.2.1.15 **qapi\_FS\_Status\_t qapi\_FS\_Iter\_Open ( const char \* *Path*, qapi\_FS\_Iter\_Handle\_t \* *handle* )**

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.

#### Parameters

in	<i>Path</i>	Valid path of the directory to iterate.
out	<i>handle</i>	Handle provided by the module to the client.

#### Note

Clients should cache the handle. Once lost, it cannot be queried back from the module.

#### Returns

Returns QAPI\_OK on success and -ve error code is returned on failure.

### 13.2.1.16 **qapi\_FS\_Status\_t qapi\_FS\_Iter\_Next ( qapi\_FS\_Iter\_Handle\_t *Iter\_Hdl*, struct qapi\_FS\_Iter\_Entry\_s \* *Iter\_Entry* )**

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.

**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 <a href="#">qapi_FS_Iter_Open()</a> API.
out	<i>Iter_Entry</i>	Details about the next entry found is filled in struct <a href="#">qapi_FS_Dirent</a> { char file_Path[QAPI_FS_NAME_MAX+1] struct <a href="#">qapi_FS_Stat_Type_s</a> SBuf uint32 <a href="#">qapi_FS_D_Stats_Present</a> ; }

- 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.

#### 13.2.1.17 [qapi\\_FS\\_Status\\_t](#) [qapi\\_FS\\_Iter\\_Close](#) ( [qapi\\_FS\\_Iter\\_Handle\\_t](#) *Iter\_Hdl* )

Closes the directory iterator, releasing the iterator for reuse.

### Parameters

in	<i>Iter_Hdl</i>	Handle to the directory obtained using the <a href="#">qapi_FS_Iter_Open()</a> API.
----	-----------------	---

### Returns

Returns QAPI\_OK on success and -ve error code is returned on failure.



### 13.2.1.18 qapi\_FS\_Status\_t qapi\_FS\_Last\_Error ( void )

Returns the last error that occurred in current task's 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.

#### Returns

Returns QAPI\_OK on success and -ve error code is returned on failure.

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@qcom.com

## 13.3 FTL Data Types and APIs

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.

### 13.3.1 Data Structure Documentation

#### 13.3.1.1 struct qapi\_FTL\_info

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.

### 13.3.2 Typedef Documentation

#### 13.3.2.1 typedef struct qapi\_FTL\_info qapi\_FTL\_info\_t

Structure for storing information about a partition.

#### 13.3.2.2 typedef void\* qapi\_FTL\_client\_t

Handle returned to the client. One handle is returned per partition.

### 13.3.3 Function Documentation

#### 13.3.3.1 qapi\_Status\_t qapi\_FTL\_Open ( qapi\_FTL\_client\_t \* *handle*, const uint8\_t \* *part\_name* )

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 exposed 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.

**Note:** One handle is returned per partition.

#### 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 FLASH_FTL_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

- FLASH\_FTL\_INVALID\_PARAM – handle or *part\_name* is NULL, or *part\_name* is invalid.
- FLASH\_FTL\_FAIL – An internal failure occurred.
- FLASH\_FTL\_OUT\_OF\_GOOD\_BLOCKS – The partition is not usable.
- FLASH\_FTL\_OK – Success.

#### 13.3.3.2 `qapi_Status_t qapi_FTL_Close ( qapi_FTL_client_t * handle )`

Closes a partition once the client is done with it.

#### Parameters

in	<i>handle</i>	Handle of the partition to be closed.
----	---------------	---------------------------------------

#### Returns

- FLASH\_FTL\_INVALID\_PARAM – handle or *part\_name* is NULL, or *part\_name* is invalid.
- FLASH\_FTL\_FAIL – An internal failure occurred.
- FLASH\_FTL\_OK – Success.

#### 13.3.3.3 `qapi_Status_t qapi_FTL_Get_info ( qapi_FTL_client_t handle, qapi_FTL_info_t * 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.

**Note:** The total usable partition size in kB = *lpa\_size\_in\_kbytes*\**lpa\_count*.

#### Parameters

in	<i>handle</i>	Handle returned from <code>qapi_FTL_Open()</code> .
out	<i>info</i>	Pointer to where the information is stored.

**Returns**

- FLASH\_FTL\_INVALID\_PARAM – handle or info is NULL.
- FLASH\_FTL\_OK – Success.

### 13.3.3.4 **qapi\_Status\_t qapi\_FTL\_Read\_lpa ( qapi\_FTL\_client\_t *handle*, uint32\_t *lpa*, uint32\_t *lpa\_count*, uint8\_t \* *buffer* )**

Reads data in multiples of LPA(s) or pages.

**Parameters**

in	<i>handle</i>	Handle returned from <a href="#">qapi_FTL_Open()</a> .
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.

**Returns**

- FLASH\_FTL\_INVALID\_PARAM – handle or part\_name is NULL.
- FLASH\_FTL\_FAIL – An internal failure occurred.
- FLASH\_FTL\_OUT\_OF\_GOOD\_BLOCKS – The partition is not usable.
- FLASH\_FTL\_OK – Success.

### 13.3.3.5 **qapi\_Status\_t qapi\_FTL\_Write\_lpa ( qapi\_FTL\_client\_t *handle*, uint32\_t *lpa*, uint32\_t *lpa\_count*, uint8\_t \* *buffer* )**

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.

**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.

### Parameters

in	<i>handle</i>	Handle returned from <a href="#">qapi_FTL_Open()</a> .
in	<i>lpa</i>	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.

### Returns

- FLASH\_FTL\_INVALID\_PARAM – handle or part\_name is NULL.
- FLASH\_FTL\_FAIL – An internal failure occurred.
- FLASH\_FTL\_OUT\_OF\_GOOD\_BLOCKS – The partition is not usable.
- FLASH\_FTL\_OK – Success.

#### 13.3.3.6 **qapi\_Status\_t qapi\_FTL\_Erase\_block ( qapi\_FTL\_client\_t *handle*, uint32\_t *erase\_block*, uint32\_t *erase\_block\_count* )**

Erases a block.

The block size is defined by *erase\_block\_size\_in\_kbytes*. 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 corresponds to must be erased with this API.

### Parameters

in	<i>handle</i>	Handle returned from <a href="#">qapi_FTL_Open()</a> .
in	<i>erase_block</i>	Start erase block.
in	<i>erase_block_count</i>	Number of blocks to be erased from Flash starting at <i>erase_block</i> .

### Returns

- FLASH\_FTL\_INVALID\_PARAM – handle is NULL.
- FLASH\_FTL\_FAIL – An internal failure occurred.
- FLASH\_FTL\_OUT\_OF\_GOOD\_BLOCKS – The partition is not usable.
- FLASH\_FTL\_OK – Success.

# 14 Wired Connectivity Module

---

This chapter describes the USB data types and APIs.

- [USB Data Types](#)
- [USB APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

## 14.1 USB Data Types

Type definitions for USB QAPIs.

### 14.1.1 Data Structure Documentation

#### 14.1.1.1 union qapi\_USB\_ioctl\_Param

IOCTL parameter type.

Data fields

Type	Parameter	Description
<a href="#">qapi_USB_App_Rx_Cb_t</a>	qapi_USB_App_Rx_Cb_Func	Client callback function.

### 14.1.2 Typedef Documentation

#### 14.1.2.1 typedef void(\* qapi\_USB\_App\_Rx\_Cb\_t)(void)

Client callback function type to be called when data is received from the client.

#### 14.1.2.2 typedef union qapi\_USB\_ioctl\_Param qapi\_USB\_ioctl\_Param\_t

IOCTL parameter type.

### 14.1.3 Enumeration Type Documentation

#### 14.1.3.1 enum qapi\_USB\_ioctl\_Cmd\_t

IOCTL command type.

Enumerator:

**QAPI\_USB\_RX\_CB\_REG\_E** IOCTL command argument to register a client callback.

## 14.2 USB APIs

These USB APIs enable clients to open a USB channel to allow data transfers between the client and the device without a specific packet format.

\* The code snippet below demonstrates use of this interface. The example  
 \* below opens a USB channel and then the write API helps the client send  
 \* data over USB. The Read API enables clients to get data over USB.  
 \* The client must define a callback function that is called whenever  
 \* there is data for the client, and then the client can call the Read  
 \* function.

```
void* Buffer
uint16 Max_Size
void* Data_Ptr
uint16 Len
void Callback_func(void);

// To open a USB channel
status = qapi_USB_Open();
if (status != QAPI_OK) { ... }

// To read data over USB; buffer to get data and max size it can take
status = qapi_USB_Read(&Buffer, Max_Size);
if (status != QAPI_OK) { ... }

// To send data over USB; pointer to data and length of data
status=qapi_USB_Write(Data_Ptr, Len);
if (status != QAPI_OK) { ... }

// To register a client callback
status = qapi_USB_Ioctl(QAPI_USB_RX_CB_REG_E, Callback_func);
if (status != QAPI_OK) { ... }
```

### 14.2.1 Function Documentation

#### 14.2.1.1 qapi\_USB\_Status\_t qapi\_USB\_Open ( void )

Opens a ser3 channel for pure data transfer through USB.

This channel enables a data transfer path for clients without any protocol.

##### Returns

QAPI\_OK on success, a -ve error code on failure.  
 QAPI\_ERR\_\_ALREADY\_DONE – The ser3 channel is already open.

#### 14.2.1.2 qapi\_USB\_Status\_t qapi\_USB\_Read ( void \*\* *Buffer*, uint16 *Max\_Size* )

Reads USB data.

This function is to be called after USB sends a callback that the PC has sent data. It can also be called without receiving the callback, but data might not be available with the USB.



**Parameters**

out	<i>Buffer</i>	Buffer to where the data is to be copied.
in	<i>Max_Size</i>	Maximum size of the data that the client can take.

**Returns**

QAPI\_OK on success, a -ve error code on failure.

QAPI\_ERR\_NO\_DATA – No data is available.

**14.2.1.3 qapi\_USB\_Status\_t qapi\_USB\_Write ( void \* *Data\_Ptr*, uint16 *Len* )**

Sends data over USB.

The client must send a data pointer and the length of the data it wishes to send over the channel.

**Parameters**

in	<i>Data_Ptr</i>	Pointer to the data that the client wishes to send.
in	<i>Len</i>	Length of the data to be sent.

**Returns**

QAPI\_OK on success, a -ve error code on failure.

**14.2.1.4 qapi\_USB\_Status\_t qapi\_USB\_ioctl ( qapi\_USB\_ioctl\_Cmd\_t *Cmd*, qapi\_USB\_ioctl\_Param\_t \* *Param* )**

IOCTL for registering the client Rx callback.

This IOCTL is made generic so that it may later be used for some other purposes.

**Parameters**

in	<i>Cmd</i>	Determines for what the IOCTL is called. Currently, only the purpose stated above is valid.
in	<i>Param</i>	Can change based on the command passed. For command APP_RX_CB_REG, it is a function pointer.

**Returns**

QAPI\_OK on success, a -ve error code on failure.

QAPI\_ERR\_INVALID\_PARAM – The command received is not supported.

# 15 Buses Module

---

This chapter describes the I2C, SPI, and UART APIs.

- [I2C Master APIs](#)
- [SPI Master APIs](#)
- [UART APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 15.1 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.

```
//
// The code sample below demonstrates the use of this interface.
//

void sample (void)
{
    void *client_handle = NULL;
    uint32_t transferred1, transferred2;
    uint8_t buffer[4] = { 1, 2, 3, 4 };

    qapi_Status_t res = QAPI_OK;
    qapi_I2CM_Config_t config;
    qapi_I2CM_Descriptor_t desc[2];

    // Obtain a client specific connection handle to the i2c bus instance 1
    res = qapi_I2CM_Open (QAPI_I2CM_INSTANCE_001_E, &client_handle);

    // Configure the bus speed and slave address
    config.bus_Frequency_KHz = 400;
    config.slave_Address      = 0x36;
    config.SMBUS_Mode         = FALSE;

    // <S> - START bit
    // <P> - STOP bit
    // <Sr> - Repeat Start bit
    // <A> - Acknowledge bit
    // <N> - Not-Acknowledge bit
    // <R> - Read Transfer
    // <W> - Write Transfer

    // Single write transfer of N bytes
    // <S><slave_address><W><A><data1><A><data2><A>...<dataN><A><P>
    desc[0].buffer      = buffer;
    desc[0].length      = 4;
    desc[0].transferred = &transferred1;
    desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE |
        QAPI_I2C_FLAG_STOP;
    res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 1,
        client_callback, NULL);

    // Single read transfer of N bytes
    // <S><slave_address><R><A><data1><A><data2><A>...<dataN><N><P>
    desc[0].buffer      = buffer;
    desc[0].length      = 4;
    desc[0].transferred = &transferred1;
    desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ |
        QAPI_I2C_FLAG_STOP;
    res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 1,
        client_callback, NULL);
}
```

```

// Read N bytes from a register 0x01 on a sensor device
// <S><slave_address><W><A><0x01><A><S><slave_address><R><A>
//                               <data1><A><data2><A>...<dataN><N><P>
uint8_t reg = 0x01;
desc[0].buffer      = &reg;
desc[0].length      = 1;
desc[0].transferred = &transferred1;
desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE;

desc[1].buffer      = buffer;
desc[1].length      = 4;
desc[1].transferred = &transferred2;
desc[1].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ |
    QAPI_I2C_FLAG_STOP;
res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 2,
    client_callback, NULL);

// Read N bytes from eeprom address 0x0102
// <S><slave_address><W><A><0x01><A><0x02><A><S><slave_address><R><A>
//                               <data1><A><data2><A>...<dataN><N><P>
uint8_t reg[2] = { 0x01, 0x02 };
desc[0].buffer      = reg;
desc[0].length      = 2;
desc[0].transferred = &transferred1;
desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE;

desc[1].buffer      = buffer;
desc[1].length      = 4;
desc[1].transferred = &transferred2;
desc[1].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ |
    QAPI_I2C_FLAG_STOP;
res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 2,
    client_callback, NULL);

// Close the connection handle to the i2c bus instance
res = qapi_I2CM_Close (client_handle);
}

void client_callback (uint32_t status, void *ctxt)
{
    // Transfer completed
}

```

## 15.1.1 Define Documentation

### 15.1.1.1 #define QAPI\_I2C\_FLAG\_START 0x00000001

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

### 15.1.1.2 #define QAPI\_I2C\_FLAG\_STOP 0x00000002

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

**15.1.1.3 #define QAPI\_I2C\_FLAG\_WRITE 0x00000004**

Must be set to indicate a WRITE transfer.

**15.1.1.4 #define QAPI\_I2C\_FLAG\_READ 0x00000008**

Must be set to indicate a READ transfer.

**15.1.1.5 #define QAPI\_I2C\_TRANSFER\_MASK (QAPI\_I2C\_FLAG\_WRITE | QAPI\_I2C\_FLAG\_READ)**

Transfer types.

**15.1.1.6 #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.

**15.1.2 Data Structure Documentation****15.1.2.1 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.

**15.1.2.2 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.

Type	Parameter	Description
uint32_t	transferred	Number of bytes actually transferred.
uint32_t	flags	I2C flags for the transfer.

### 15.1.3 Typedef Documentation

#### 15.1.3.1 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 <a href="#">qapi_I2CM_Get_QStatus_Code()</a> will convert this status to QAPI status codes.
out	<i>CB_Parameter</i>	CP_Parameter context that was passed in the call to <a href="#">qapi_I2CM_Transfer()</a> .

### 15.1.4 Enumeration Type Documentation

#### 15.1.4.1 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\_001\_E** I2C core 01.  
**QAPI\_I2CM\_INSTANCE\_002\_E** I2C core 02.  
**QAPI\_I2CM\_INSTANCE\_003\_E** I2C core 03.  
**QAPI\_I2CM\_INSTANCE\_004\_E** I2C core 04.  
**QAPI\_I2CM\_INSTANCE\_005\_E** I2C core 05.  
**QAPI\_I2CM\_INSTANCE\_006\_E** I2C core 06.  
**QAPI\_I2CM\_INSTANCE\_007\_E** I2C core 07.  
**QAPI\_I2CM\_INSTANCE\_008\_E** I2C core 08.  
**QAPI\_I2CM\_INSTANCE\_009\_E** I2C core 09.  
**QAPI\_I2CM\_INSTANCE\_010\_E** I2C core 10.  
**QAPI\_I2CM\_INSTANCE\_011\_E** I2C core 11.  
**QAPI\_I2CM\_INSTANCE\_012\_E** I2C core 12.  
**QAPI\_I2CM\_INSTANCE\_013\_E** I2C core 13.

**QAPI\_I2CM\_INSTANCE\_014\_E** I2C core 14.  
**QAPI\_I2CM\_INSTANCE\_015\_E** I2C core 15.  
**QAPI\_I2CM\_INSTANCE\_016\_E** I2C core 16.  
**QAPI\_I2CM\_INSTANCE\_017\_E** I2C core 17.  
**QAPI\_I2CM\_INSTANCE\_018\_E** I2C core 18.  
**QAPI\_I2CM\_INSTANCE\_019\_E** I2C core 19.  
**QAPI\_I2CM\_INSTANCE\_020\_E** I2C core 20.  
**QAPI\_I2CM\_INSTANCE\_021\_E** I2C core 21.  
**QAPI\_I2CM\_INSTANCE\_022\_E** I2C core 22.  
**QAPI\_I2CM\_INSTANCE\_023\_E** I2C core 23.  
**QAPI\_I2CM\_INSTANCE\_024\_E** I2C core 24.

## 15.1.5 Function Documentation

### 15.1.5.1 `qapi_Status_t qapi_I2CM_Open ( qapi_I2CM_Instance_t instance, void ** i2c_Handle )`

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 HW instance. To disable the power to the instance, a corresponding call to [qapi\\_I2CM\\_Close\(\)](#) must be made.

#### Parameters

in	<i>instance</i>	I2C instance that the client intends to initialize; see <a href="#">qapi_I2CM_Instance_t</a> 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\_INVALID\_PARAMETER** – Invalid parameter.
- **QAPI\_I2CM\_ERR\_API\_INVALID\_EXECUTION\_LEVEL** – Invalid execution level.
- **QAPI\_I2CM\_ERR\_UNSUPPORTED\_CORE\_INSTANCE** – Unsupported core instance.
- **QAPI\_I2CM\_ERR\_HANDLE\_ALLOCATION** – Handle allocation error.
- **QAPI\_I2CM\_ERR\_HW\_INFO\_ALLOCATION** – Hardware information allocation error.
- **QAPI\_I2CM\_ERR\_PLATFORM\_INIT\_FAIL** – Platform initialization failure.
- **QAPI\_I2CM\_ERR\_PLATFORM\_REG\_INT\_FAIL** – Platform registration internal failure.
- **QAPI\_I2CM\_ERR\_PLATFORM\_CLOCK\_ENABLE\_FAIL** – Platform clock enable failure.
- **QAPI\_I2CM\_ERR\_PLATFORM\_GPIO\_ENABLE\_FAIL** – Platform GPIO enable failure.

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

De-initializes the I2C instance and releases any resources allocated by the [qapi\\_I2CM\\_Open\(\)](#) API.

#### Parameters

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

#### Returns

- QAPI\_OK – Function was successful.
- QAPI\_I2CM\_ERR\_INVALID\_PARAMETER – Invalid parameter.
- QAPI\_I2CM\_ERR\_API\_INVALID\_EXECUTION\_LEVEL – Invalid execution level.
- QAPI\_I2CM\_ERR\_PLATFORM\_DEINIT\_FAIL – Platform de-initialization failure.
- QAPI\_I2CM\_ERR\_PLATFORM\_DE\_REG\_INT\_FAIL – Platform de-registration internal failure.
- QAPI\_I2CM\_ERR\_PLATFORM\_CLOCK\_DISABLE\_FAIL – Platform clock disable failure.
- QAPI\_I2CM\_ERR\_PLATFORM\_GPIO\_DISABLE\_FAIL – Platform GPIO disable failure.

### 15.1.5.3 `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 )`

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.

#### Parameters

in	<i>i2c_Handle</i>	Handle to the I2C instance.
in	<i>config</i>	Slave configuration. See <a href="#">qapi_I2CM_Config_t</a> for details.
in	<i>desc</i>	I2C transfer descriptor. See <a href="#">qapi_I2CM_Descriptor_t</a> 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>	Delay in microseconds.
----	-----------------	------------------------

**Returns**

- QAPI\_OK – Function was successful.
- QAPI\_I2CM\_ERR\_INVALID\_PARAMETER – Invalid parameter.
- QAPI\_I2CM\_ERR\_API\_INVALID\_EXECUTION\_LEVEL – Invalid execution level.
- QAPI\_I2CM\_ERR\_TRANSFER\_TIMEOUT – Transfer timed out.
- QAPI\_I2CM\_ERR\_QSTATE\_INVALID – QState is invalid.
- QAPI\_I2CM\_ERROR\_HANDLE\_ALREADY\_IN\_QUEUE – Client IO is pending.

**15.1.5.4 qapi\_Status\_t qapi\_I2CM\_Power\_On ( void \* *i2c\_Handle* )**

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 <a href="#">qapi_I2CM_Open()</a> .
----	-------------------	--

**Returns**

- QAPI\_OK – I2C master enabled successfully.
- QAPI\_I2CM\_ERROR\_INVALID\_PARAM – Invalid handle parameter.
- QAPI\_I2CM\_ERROR\_CLK\_ENABLE\_FAIL – Could not enable clocks or NPA.
- QAPI\_I2CM\_ERROR\_GPIO\_ENABLE\_FAIL – Could not enable GPIOs.

**15.1.5.5 qapi\_Status\_t qapi\_I2CM\_Power\_Off ( void \* *i2c\_Handle* )**

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.

**Parameters**

in	<i>i2c_Handle</i>	Driver handle returned by <a href="#">qapi_I2CM_Open()</a> .
----	-------------------	--

**Returns**

- QAPI\_OK – I2C master disabled successfully.
- QAPI\_I2CM\_ERROR\_INVALID\_PARAM – Invalid handle parameter.

- QAPI\_I2CM\_ERROR\_CLK\_DISABLE\_FAIL – Could not disable clocks or NPA.
- QAPI\_I2CM\_ERROR\_GPIO\_DISABLE\_FAIL – Could not disable GPIOs.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@quectel.com

## 15.2 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

### 15.2.1 Data Structure Documentation

#### 15.2.1.1 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
<a href="#">qapi_SPIM_Shift_Mode_t</a>	SPIM_Mode	SPIM mode type to be used for the SPI core.
<a href="#">qapi_SPIM_CS_Polarity_t</a>	SPIM_CS_Polarity	CS polarity type to be used for the SPI core.
<a href="#">qapi_SPIM_Byte_Order_t</a>	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_Cycles	Number of clock cycles to wait after asserting CS before starting transfer.
uint8_t	Inter_Word_Delay_Cycles	Number of clock cycles to wait between SPI words.
<a href="#">qapi_SPIM_CS_Mode_t</a>	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.

**15.2.1.2 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.

**15.2.2 Typedef Documentation****15.2.2.1 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.

## 15.2.3 Enumeration Type Documentation

### 15.2.3.1 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** SPIM instance 1.  
**QAPI\_SPIM\_INSTANCE\_2\_E** SPIM instance 2.  
**QAPI\_SPIM\_INSTANCE\_3\_E** SPIM instance 3.  
**QAPI\_SPIM\_INSTANCE\_4\_E** SPIM instance 4.  
**QAPI\_SPIM\_INSTANCE\_5\_E** SPIM instance 5.  
**QAPI\_SPIM\_INSTANCE\_6\_E** SPIM instance 6.  
**QAPI\_SPIM\_INSTANCE\_7\_E** SPIM instance 7.  
**QAPI\_SPIM\_INSTANCE\_8\_E** SPIM instance 8.  
**QAPI\_SPIM\_INSTANCE\_9\_E** SPIM instance 9.  
**QAPI\_SPIM\_INSTANCE\_10\_E** SPIM instance 10.  
**QAPI\_SPIM\_INSTANCE\_11\_E** SPIM instance 11.  
**QAPI\_SPIM\_INSTANCE\_12\_E** SPIM instance 12.  
**QAPI\_SPIM\_INSTANCE\_13\_E** SPIM instance 13.  
**QAPI\_SPIM\_INSTANCE\_14\_E** SPIM instance 14.  
**QAPI\_SPIM\_INSTANCE\_15\_E** SPIM instance 15.  
**QAPI\_SPIM\_INSTANCE\_16\_E** SPIM instance 16.  
**QAPI\_SPIM\_INSTANCE\_17\_E** SPIM instance 17.  
**QAPI\_SPIM\_INSTANCE\_18\_E** SPIM instance 18.  
**QAPI\_SPIM\_INSTANCE\_19\_E** SPIM instance 19.  
**QAPI\_SPIM\_INSTANCE\_20\_E** SPIM instance 20.  
**QAPI\_SPIM\_INSTANCE\_21\_E** SPIM instance 21.  
**QAPI\_SPIM\_INSTANCE\_22\_E** SPIM instance 22.  
**QAPI\_SPIM\_INSTANCE\_23\_E** SPIM instance 23.  
**QAPI\_SPIM\_INSTANCE\_24\_E** SPIM instance 24.

### 15.2.3.2 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** CPOL = 0, CPHA = 0.  
**QAPI\_SPIM\_MODE\_1\_E** CPOL = 0, CPHA = 1.  
**QAPI\_SPIM\_MODE\_2\_E** CPOL = 1, CPHA = 0.  
**QAPI\_SPIM\_MODE\_3\_E** CPOL = 1, CPHA = 1.

### 15.2.3.3 enum qapi\_SPIM\_CS\_Polarity\_t

SPI chip select polarity type.

Enumerator:

**QAPI\_SPIM\_CS\_ACTIVE\_LOW\_E** During Idle state, the CS line is held low.

**QAPI\_SPIM\_CS\_ACTIVE\_HIGH\_E** During Idle state, the CS line is held high.

### 15.2.3.4 enum qapi\_SPIM\_Byte\_Order\_t

Order in which bytes from Tx/Rx buffer words are put on the bus.

Enumerator:

**SPI\_NATIVE** Native.

**SPI\_LITTLE\_ENDIAN** Little Endian.

**SPI\_BIG\_ENDIAN** Big Endian (network).

### 15.2.3.5 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** CS is deasserted after transferring data for N clock cycles.

**QAPI\_SPIM\_CS\_KEEP\_ASSERTED\_E** CS is asserted as long as the core is in the Run state.

## 15.2.4 Function Documentation

### 15.2.4.1 `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.

#### Parameters

in	<i>instance</i>	SPI instance specified by <a href="#">qapi_SPIM_Instance_t</a> .
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\_MEM\_ALLOC – Could not allocate space for driver structures.
- QAPI\_SPIM\_ERR\_INTERRUPT\_REGISTER – Could not register for an interrupt.

### 15.2.4.2 `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.

#### Parameters

in	<i>spi_Handle</i>	Driver handle returned by <a href="#">qapi_SPIM_Open()</a> .
----	-------------------	--

#### Returns

- QAPI\_OK – SPI master enabled successfully.
- QAPI\_SPIM\_ERROR\_INVALID\_PARAM – Invalid handle parameter.
- QAPI\_SPIM\_ERROR\_CLK\_ENABLE\_FAIL – Could not enable clocks or NPA.
- QAPI\_SPIM\_ERROR\_GPIO\_ENABLE\_FAIL – Could not enable GPIOs.

### 15.2.4.3 **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.

#### Parameters

in	<i>spi_Handle</i>	Driver handle returned by <a href="#">qapi_SPIM_Open()</a> .
----	-------------------	--

#### Returns

- QAPI\_OK – SPI master disabled successfully.
- QAPI\_SPIM\_ERROR\_INVALID\_PARAM – Invalid handle parameter.
- QAPI\_SPIM\_ERROR\_CLK\_DISABLE\_FAIL – Could not disable clocks or NPA.
- QAPI\_SPIM\_ERROR\_GPIO\_DISABLE\_FAIL – Could not disable GPIOs.

### 15.2.4.4 **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.

#### Parameters

in	<i>spi_Handle</i>	Driver handle returned by <a href="#">qapi_SPIM_Open()</a> .
in	<i>config</i>	Pointer to the SPI configuration structure described by <a href="#">qapi_SPIM_Config_t</a> .
in	<i>desc</i>	Pointer to the structure described by <a href="#">qapi_SPIM_Descriptor_t</a> . The descriptor can have NULL Tx OR Tx buffers if a half duplex transfer is selected.
in	<i>num_Descriptors</i>	Number of descriptor pointers the client wants to process.
in	<i>c_Fn</i>	Callback function to be invoked when the SPI transfer completes successfully or with an error.
in	<i>c_Ctxt</i>	Pointer to a client object that will be returned as an argument to <i>c_Fn</i> .
in	<i>get_timestamp</i>	Timestamp of the data transfer.

#### Returns

- QAPI\_OK – SPI master enabled successfully.
- QAPI\_SPIM\_ERROR\_INVALID\_PARAM – One or more invalid parameters.



- QAPI\_SPIM\_ERROR\_QUP\_STATE\_INVALID – SPI or BAM hardware is in a bad state.
- QAPI\_SPIM\_ERROR\_TRANSFER\_TIMEOUT – Transfer timed out.

#### 15.2.4.5 `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.

##### Parameters

in	<i>spi_handle</i>	Driver handle returned by <a href="#">qapi_SPIM_Open()</a> .
----	-------------------	--

##### Returns

- QAPI\_OK – SPI driver closed successfully.

## 15.3 UART APIs

This section describes the UART data types and APIs.

### 15.3.1 Data Structure Documentation

#### 15.3.1.1 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.
<a href="#">QAPI_Flow_Control_Type</a>	Flow_Control_Type	Transmit flow control type.

#### 15.3.1.2 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.
<a href="#">qapi_UART_Parity_Mode_e</a>	parity_Mode	Parity mode.
<a href="#">qapi_UART_Num_Stop_Bits_e</a>	num_Stop_Bits	Number of stop bits.
<a href="#">qapi_UART_Bits_Per_Char_e</a>	bits_Per_Char	Bits per character.
qbool_t	enable_Loopback	Enable loopback.
qbool_t	enable_Flow_Ctrl	Enable flow control.
<a href="#">qapi_UART_Callback_Fn_t</a>	tx_CB_ISR	Transmit callback, called from ISR context. Be sure not to violate ISR guidelines. <b>Note:</b> Do not call uart_transmit or uart_receive APIs from this callback.
<a href="#">qapi_UART_Callback_Fn_t</a>	rx_CB_ISR	Receive callback, called from ISR context. Be sure not to violate ISR guidelines. <b>Note:</b> Do not call uart_transmit or uart_receive APIs from this callback.

## 15.3.2 Typedef Documentation

### 15.3.2.1 typedef union QAPI\_UART\_ioctl\_Param QAPI\_UART\_ioctl\_Param

IOCTL command ID list of the UART.

### 15.3.2.2 typedef void\* qapi\_UART\_Handle\_t

UART handle that is passed to the client when a UART port is opened.

### 15.3.2.3 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.

## 15.3.3 Enumeration Type Documentation

### 15.3.3.1 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*** UART core 01.  
***QAPI\_UART\_PORT\_002\_E*** UART core 02.  
***QAPI\_UART\_PORT\_003\_E*** UART core 03.  
***QAPI\_UART\_PORT\_004\_E*** UART core 04.  
***QAPI\_UART\_PORT\_005\_E*** UART core 05.  
***QAPI\_UART\_PORT\_006\_E*** UART core 06.  
***QAPI\_UART\_PORT\_007\_E*** UART core 07.  
***QAPI\_UART\_PORT\_008\_E*** UART core 08.  
***QAPI\_UART\_PORT\_009\_E*** UART core 09.  
***QAPI\_UART\_PORT\_010\_E*** UART core 10.  
***QAPI\_UART\_PORT\_011\_E*** UART core 11.  
***QAPI\_UART\_PORT\_012\_E*** UART core 12.  
***QAPI\_UART\_PORT\_013\_E*** UART core 13.  
***QAPI\_UART\_PORT\_014\_E*** UART core 14.  
***QAPI\_UART\_PORT\_015\_E*** UART core 15.  
***QAPI\_UART\_PORT\_016\_E*** UART core 16.  
***QAPI\_UART\_PORT\_017\_E*** UART core 17.  
***QAPI\_UART\_PORT\_018\_E*** UART core 18.  
***QAPI\_UART\_PORT\_019\_E*** UART core 19.  
***QAPI\_UART\_PORT\_020\_E*** UART core 20.

**QAPI\_UART\_PORT\_021\_E** UART core 21.  
**QAPI\_UART\_PORT\_022\_E** UART core 22.  
**QAPI\_UART\_PORT\_023\_E** UART core 23.  
**QAPI\_UART\_PORT\_024\_E** UART core 24.

### 15.3.3.2 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** 5 bits per character.  
**QAPI\_UART\_6\_BITS\_PER\_CHAR\_E** 6 bits per character.  
**QAPI\_UART\_7\_BITS\_PER\_CHAR\_E** 7 bits per character.  
**QAPI\_UART\_8\_BITS\_PER\_CHAR\_E** 8 bits per character.

### 15.3.3.3 enum qapi\_UART\_Num\_Stop\_Bits\_e

Enumeration for UART number of stop bits configuration.

Enumerator:

**QAPI\_UART\_0\_5\_STOP\_BITS\_E** 0.5 stop bits.  
**QAPI\_UART\_1\_0\_STOP\_BITS\_E** 1.0 stop bit.  
**QAPI\_UART\_1\_5\_STOP\_BITS\_E** 1.5 stop bits.  
**QAPI\_UART\_2\_0\_STOP\_BITS\_E** 2.0 stop bits.

### 15.3.3.4 enum qapi\_UART\_Parity\_Mode\_e

Enumeration for UART parity mode configuration.

Enumerator:

**QAPI\_UART\_NO\_PARITY\_E** No parity.  
**QAPI\_UART\_ODD\_PARITY\_E** Odd parity.  
**QAPI\_UART\_EVEN\_PARITY\_E** Even parity.  
**QAPI\_UART\_SPACE\_PARITY\_E** Space parity.

### 15.3.3.5 enum qapi\_UART\_ioctl\_Command\_e

IOCTL command ID list of the UART.

Enumerator:

**QAPI\_SET\_FLOW\_CTRL\_E** Set auto flow control.  
**QAPI\_SET\_BAUD\_RATE\_E** Set baud rate.

### 15.3.3.6 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.

## 15.3.4 Function Documentation

### 15.3.4.1 qapi\_Status\_t qapi\_UART\_Close ( qapi\_UART\_Handle\_t *handle* )

Closes the UART port.

Releases clock, interrupt, and GPIO handles related to this UART and cancels any pending transfers.

**Note:** Do not call this API from ISR context.

**Parameters**

in	<i>handle</i>	UART handle provided by <a href="#">qapi_UART_Open()</a> .
----	---------------	--

**Returns**

- QAPI\_OK Port close successful.
- QAPI\_ERROR Port close failed.

### 15.3.4.2 qapi\_Status\_t qapi\_UART\_Open ( qapi\_UART\_Handle\_t \* *handle*, qapi\_UART\_Port\_Id\_e *id*, qapi\_UART\_Open\_Config\_t \* *config* )

Initializes the UART port.

Opens the UART port and configures the corresponding clocks, interrupts, and GPIO.

**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 successful.
- QAPI\_ERROR Port open failed.

### 15.3.4.3 **qapi\_Status\_t qapi\_UART\_Receive ( qapi\_UART\_Handle\_t *handle*, char \* *buf*, uint32\_t *buf\_Size*, void \* *cb\_Data* )**

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.

**Note:** Do not call this API from ISR context.

#### Parameters

in	<i>handle</i>	UART handle provided by <a href="#">qapi_UART_Open()</a> .
in	<i>buf</i>	Buffer to be filled with data.
in	<i>buf_Size</i>	Buffer size. Must be $\geq 4$ 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.

### 15.3.4.4 **qapi\_Status\_t qapi\_UART\_Transmit ( qapi\_UART\_Handle\_t *handle*, char \* *buf*, uint32\_t *bytes\_To\_Tx*, void \* *cb\_Data* )**

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.

#### Parameters

in	<i>handle</i>	UART handle provided by <a href="#">qapi_UART_Open()</a> .
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.

**15.3.4.5 qapi\_Status\_t qapi\_UART\_Power\_On ( qapi\_UART\_Handle\_t UART\_Handle )**

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.

**Parameters**

in	UART_Handle	Driver handle returned by <a href="#">qapi_UART_Open()</a> .
----	-------------	--

**Returns**

- QAPI\_OK UART powered on successfully.
- QAPI\_ERROR UART power on is failed.

**15.3.4.6 qapi\_Status\_t qapi\_UART\_Power\_Off ( qapi\_UART\_Handle\_t UART\_Handle )**

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.

**Parameters**

in	UART_Handle	Driver handle returned by <a href="#">qapi_UART_Open()</a> .
----	-------------	--

**Returns**

- QAPI\_OK UART powered off successfully.
- QAPI\_ERROR UART power off is failed.

**15.3.4.7 qapi\_Status\_t qapi\_UART\_Ioctl ( qapi\_UART\_Handle\_t handle, qapi\_UART\_Ioctl\_Command\_e ioctl\_Command, void \* ioctl\_Param )**

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.

**Parameters**

in	<i>handle</i>	Driver handle returned by <a href="#">qapi_UART_Open()</a> .
in	<i>ioctl_Command</i>	Pass the commands listed with <a href="#">qapi_UART_Ioctl_Command_e</a> .
in	<i>ioctl_Param</i>	Pass the argument associated with <a href="#">qapi_UART_Ioctl_Command_e</a> .

**Returns**

- QAPI\_OK UART IOCTL configuration is successfull.
- QAPI\_ERROR UART IOCTL configuration is failed.

QUALCOMM  
2017-05-07 17:57:37 PDT  
hyman.ding@quectel.com



# 16 Location Module

---

This chapter describes the data types and APIs for the GNSS location driver.

- [Location APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

## 16.1 Location APIs

This section describes data types and functions for the GNSS location driver.

### 16.1.1 Data Structure Documentation

#### 16.1.1.1 struct qapi\_Location\_Options\_t

Structure for location options.

##### Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of LocationOptions.
uint32_t	minInterval	Minimum interval in milliseconds.
uint32_t	minDistance	Minimum distance in meters.

#### 16.1.1.2 struct qapi\_Geofence\_Options\_t

Structure for Geofence options.

##### Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of GeofenceOptions.
uint8_t	breachType-Mask	Bitwise OR of GeofenceBreachTypeMask bits.
uint32_t	responsiveness	Responsiveness in milliseconds.
uint32_t	dwellTime	Dwell time in seconds.

#### 16.1.1.3 struct qapi\_Geofence\_Data\_t

Structure for Geofence data.

##### Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of GeofenceData.
double	latitude	Latitude in degrees.
double	longitude	Longitude in degrees.
double	radius	Radius in meters.

#### 16.1.1.4 struct qapi\_Location\_t

Structure for location information.

**Data fields**

Type	Parameter	Description
size_t	size	Size. Set to the size of Location.
uint64_t	timestamp	UTC timestamp for a location fix; milliseconds since Jan. 1, 1970.
double	latitude	Latitude in degrees.
double	longitude	Longitude in degrees.
float	altitude	Altitude in meters above the WGS 84 reference ellipsoid.
float	speed	Speed in meters per second.
float	bearing	Bearing in degrees; range: 0 to 360.
float	accuracy	Accuracy in meters.
uint8_t	flags	Bitwise OR of LocationFlags.

**16.1.1.5 struct qapi\_Location\_Callbacks\_t**

Location callbacks requirements.

**Data fields**

Type	Parameter	Description
size_t	size	Size. Set to the size of <a href="#">qapi_Location_Callbacks_t</a> .
<a href="#">qapi_Capabilities_Callback</a>	capabilitiesCb	Capabilities callback is mandatory.
<a href="#">qapi_Response_Callback</a>	responseCb	Response callback is mandatory.
<a href="#">qapi_Geofence_Response_Callback</a>	geofence-ResponseCb	Geofence response callback is mandatory.
<a href="#">qapi_Tracking_Callback</a>	trackingCb	Tracking callback is optional.
<a href="#">qapi_Batching_Callback</a>	batchingCb	Batching callback is optional.
<a href="#">qapi_Geofence_Breach_Callback</a>	geofence-BreachCb	Geofence breach callback is optional.

**16.1.2 Typedef Documentation****16.1.2.1 typedef void(\* qapi\_Capabilities\_Callback)(uint16\_t capabilitiesMask)**

Provides the capabilities of the system. It is called once after [qapi\\_Loc\\_Init\(\)](#) is called.

**Parameters**

in	<i>capabilitiesMask</i>	Bitwise OR of <a href="#">qapi_Location_Capabilities_Mask_t</a> .
----	-------------------------	---

**Returns**

None.

### 16.1.2.2 **typedef void(\* qapi\_Response\_Callback)(qapi\_Location\_Error\_t err, uint32\_t id)**

Response callback, which is used by tracking, batching, and miscellaneous APIs. It is called for every tracking, batching, and miscellaneous API.

**Parameters**

in	<i>err</i>	Error; if not SUCCESS, the ID is not valid.
in	<i>id</i>	ID to be associated with the request.

**Returns**

None.

### 16.1.2.3 **typedef void(\* qapi\_Geofence\_Response\_Callback)(size\_t count, qapi\_Location\_Error\_t \*err, uint32\_t \*ids)**

Geofence response callback is used by Geofence APIs. It is called for every Geofence API call.

**Parameters**

in	<i>count</i>	Number of locations in arrays.
in	<i>err</i>	Array of <a href="#">qapi_Location_Error_t</a> associated with the request.
in	<i>ids</i>	Array of IDs to be associated with the request.

**Returns**

None.

### 16.1.2.4 **typedef void(\* qapi\_Tracking\_Callback)(qapi\_Location\_t location)**

Tracking callback used for the [qapi\\_Loc\\_Start\\_Tracking\(\)](#) API. This is an optional function and can be NULL. It is called when delivering a location in a tracking session.

**Parameters**

in	<i>location</i>	Structure containing information related to the tracked location.
----	-----------------	---

**Returns**

None.

### 16.1.2.5 typedef void(\* qapi\_Batching\_Callback)(size\_t count, qapi\_Location\_t \*location)

Batching callback used for the [qapi\\_Loc\\_Start\\_Batching\(\)](#) API. This is an optional function and can be NULL. It is called when delivering a location in a batching session.

#### Parameters

in	<i>count</i>	Number of locations in an array.
in	<i>location</i>	Array of location structures containing information related to the batched locations.

#### Returns

None.

### 16.1.2.6 typedef void(\* qapi\_Geofence\_Breach\_Callback)(size\_t count, uint32\_t \*ids, qapi\_Location\_t location, qapi\_Geofence\_Breach\_t breachType, uint64\_t timestamp)

Geofence breach callback used for the [qapi\\_Loc\\_Add\\_Geofences\(\)](#) API. This is an optional function and can be NULL. It is called when any number of geofences have a state change.

#### Parameters

in	<i>count</i>	Number of IDs in the array.
in	<i>ids</i>	Array of IDs that have been breached.
in	<i>location</i>	Location associated with a breach.
in	<i>breachType</i>	Type of breach.
in	<i>timestamp</i>	Timestamp of the breach.

#### Returns

None

## 16.1.3 Enumeration Type Documentation

### 16.1.3.1 enum qapi\_Location\_Error\_t

GNSS location error codes.

#### Enumerator:

**QAPI\_LOCATION\_ERROR\_SUCCESS** Success.  
**QAPI\_LOCATION\_ERROR\_GENERAL\_FAILURE** General failure.  
**QAPI\_LOCATION\_ERROR\_CALLBACK\_MISSING** Callback is missing.  
**QAPI\_LOCATION\_ERROR\_INVALID\_PARAMETER** Invalid parameter.  
**QAPI\_LOCATION\_ERROR\_ID\_EXISTS** ID already exists.  
**QAPI\_LOCATION\_ERROR\_ID\_UNKNOWN** ID is unknown.

**QAPI\_LOCATION\_ERROR\_ALREADY\_STARTED** Already started.

**QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED** Not initialized.

### 16.1.3.2 enum qapi\_Location\_Flags\_t

Flags to indicate which values are valid in a location.

Enumerator:

**QAPI\_LOCATION\_HAS\_LAT\_LONG\_BIT** Location has a valid latitude and longitude.

**QAPI\_LOCATION\_HAS\_ALTITUDE\_BIT** Location has a valid altitude.

**QAPI\_LOCATION\_HAS\_SPEED\_BIT** Location has a valid speed.

**QAPI\_LOCATION\_HAS\_BEARING\_BIT** Location has a valid bearing.

**QAPI\_LOCATION\_HAS\_ACCURACY\_BIT** Location has valid accuracy.

### 16.1.3.3 enum qapi\_Geofence\_Breach\_t

Flags to indicate Geofence breach status.

Enumerator:

**QAPI\_GEOFENCE\_BREACH\_ENTER** Entering Geofence breach.

**QAPI\_GEOFENCE\_BREACH\_EXIT** Exiting Geofence breach.

**QAPI\_GEOFENCE\_BREACH\_DWELL\_IN** Dwelling in a breached Geofence.

**QAPI\_GEOFENCE\_BREACH\_DWELL\_OUT** Dwelling outside of a breached Geofence.

**QAPI\_GEOFENCE\_BREACH\_UNKNOWN** Breach is unknown.

### 16.1.3.4 enum qapi\_Geofence\_Breach\_Mask\_t

Flags to indicate Geofence breach mask bit.

Enumerator:

**QAPI\_GEOFENCE\_BREACH\_ENTER\_BIT** Breach enter bit.

**QAPI\_GEOFENCE\_BREACH\_EXIT\_BIT** Breach exit bit.

**QAPI\_GEOFENCE\_BREACH\_DWELL\_IN\_BIT** Breach dwell in bit.

**QAPI\_GEOFENCE\_BREACH\_DWELL\_OUT\_BIT** Breach dwell out bit.

### 16.1.3.5 enum qapi\_Location\_Capabilities\_Mask\_t

Flags to indicate the capabilities bit.

Enumerator:

**QAPI\_LOCATION\_CAPABILITIES\_TRACKING\_BIT** Capabilities tracking bit.

**QAPI\_LOCATION\_CAPABILITIES\_BATCHING\_BIT** Capabilities batching bit.

**QAPI\_LOCATION\_CAPABILITIES\_GEOFENCE\_BIT** Capabilities Geofence bit.

## 16.1.4 Function Documentation

### 16.1.4.1 `qapi_Location_Error_t qapi_Loc_Init ( qapi_Location_Callbacks_t callbacks )`

Initializes a location session and registers the callbacks.

#### Parameters

in	<i>callbacks</i>	Structure with the callback functions to be registered.
----	------------------	---

#### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_CALLBACK\_MISSING if one of the mandatory callback functions is missing.
- QAPI\_LOCATION\_ERROR\_GENERAL\_FAILURE if there is an internal error.
- QAPI\_LOCATION\_ERROR\_ALREADY\_STARTED if a location session has already been initialized.

### 16.1.4.2 `qapi_Location_Error_t qapi_Loc_Deinit ( void )`

Deinitializes a location session.

#### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

### 16.1.4.3 `qapi_Location_Error_t qapi_Loc_Start_Tracking ( qapi_Location_Options_t * pOptions, uint32_t * pSessionId )`

Starts a tracking session, which returns a session ID that will be used by the other tracking APIs and in the response callback to match the command with a response. Locations are reported on the tracking callback passed in `qapi_Loc_Init()` periodically according to the location options.

#### Parameters

in	<i>pOptions</i>	Pointer to a structure containing the options: <ul style="list-style-type: none"> <li>• minInterval – Minimum interval between fixes</li> <li>• minDistance – Minimum distance to travel before reporting</li> </ul>
in	<i>pSessionId</i>	Pointer to the session ID to be returned.

#### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.

- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.4 qapi\_Location\_Error\_t qapi\_Loc\_Stop\_Tracking ( uint32\_t *id* )

Stops a tracking session associated with the *id* parameter.

##### Parameters

in	<i>id</i>	ID of the session to be stopped.
----	-----------	----------------------------------

##### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.5 qapi\_Location\_Error\_t qapi\_Loc\_Update\_Tracking\_Options ( uint32\_t *id*, qapi\_Location\_Options\_t \* *pOptions* )

Changes the location options of a tracking session associated with the *id* parameter.

##### Parameters

in	<i>id</i>	ID of the session to be changed.
in	<i>pOptions</i>	Pointer to a structure containing the options: <ul style="list-style-type: none"> <li>• minInterval – Minimum interval between fixes</li> <li>• minDistance – Minimum distance to travel before reporting</li> </ul>

##### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.6 qapi\_Location\_Error\_t qapi\_Loc\_Start\_Batching ( qapi\_Location\_Options\_t \* *pOptions*, uint32\_t \* *pSessionId* )

Starts a batching session, which returns a session ID that will be used by the other batching APIs and in the response callback to match the command with a response.

Locations are reported on the batching callback passed in [qapi\\_Loc\\_Init\(\)](#) periodically according to the location options. A batching session starts tracking on the low power processor and delivers them in batches by the batching callback when the batch is full or when [qapi\\_Loc\\_Get\\_Batched\\_Locations\(\)](#) is called. This allows for the processor that calls this API to sleep when the low power processor can batch locations in the background and wake up the processor calling the API only when the batch is full, thus saving power.



**Parameters**

in	<i>pOptions</i>	Pointer to a structure containing the options: <ul style="list-style-type: none"> <li>• minInterval – Minimum interval between fixes</li> <li>• minDistance – Minimum distance to travel before reporting</li> </ul>
in	<i>pSessionId</i>	Pointer to the session ID to be returned.

**Returns**

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

**16.1.4.7 qapi\_Location\_Error\_t qapi\_Loc\_Stop\_Batching ( uint32\_t id )**

Stops a batching session associated with the id parameter.

**Parameters**

in	<i>id</i>	ID of the session to be stopped.
----	-----------	----------------------------------

**Returns**

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

**16.1.4.8 qapi\_Location\_Error\_t qapi\_Loc\_Update\_Batching\_Options ( uint32\_t id, qapi\_Location\_Options\_t \* pOptions )**

Changes the location options of a batching session associated with the id parameter.

**Parameters**

in	<i>id</i>	ID of the session to be changed.
in	<i>pOptions</i>	Pointer to a structure containing the options: <ul style="list-style-type: none"> <li>• minInterval – Minimum interval between fixes</li> <li>• minDistance – Minimum distance to travel before reporting</li> </ul>

**Returns**

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.9 **qapi\_Location\_Error\_t qapi\_Loc\_Get\_Batched\_Locations ( uint32\_t *id*, size\_t *count* )**

Gets a number of locations that are currently stored or batched on the low power processor, delivered by the batching callback passed to [qapi\\_Loc\\_Init\(\)](#). Locations are then deleted from the batch stored on the low power processor.

##### Parameters

in	<i>id</i>	ID of the session for which the number of locations is requested.
in	<i>count</i>	Number of requested locations.

##### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.10 **qapi\_Location\_Error\_t qapi\_Loc\_Add\_Geofences ( size\_t *count*, qapi\_Geofence\_Options\_t \* *pOptions*, qapi\_Geofence\_Data\_t \* *pData*, uint32\_t \*\* *pIdArray* )**

Adds a specified number of Geofences and returns an array of Geofence IDs that will be used by the other Geofence APIs, as well as in the Geofence response callback to match the command with a response. The Geofence breach callback delivers the status of each Geofence according to the Geofence options for each.

##### Parameters

in	<i>count</i>	Number of Geofences to be added.
in	<i>pOptions</i>	Array of structures containing the options: <ul style="list-style-type: none"> <li>• breachTypeMask – Bitwise OR of GeofenceBreachTypeMask bits</li> <li>• responsiveness in milliseconds</li> <li>• dwellTime in seconds</li> </ul>
in	<i>pData</i>	Array of structures containing the data: <ul style="list-style-type: none"> <li>• latitude of the center of the Geofence in degrees</li> <li>• longitude of the center of the Geofence in degrees</li> <li>• radius of the Geofence in meters</li> </ul>
in	<i>pIdArray</i>	Array of IDs of Geofences to be returned.

##### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.11 **qapi\_Location\_Error\_t qapi\_Loc\_Remove\_Geofences ( size\_t *count*, uint32\_t \* *pIDs* )**

Removes a specified number of Geofences.

##### Parameters

in	<i>count</i>	Number of Geofences to be removed.
in	<i>pIDs</i>	Array of IDs of the Geofences to be removed.

##### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.12 **qapi\_Location\_Error\_t qapi\_Loc\_Modify\_Geofences ( size\_t *count*, uint32\_t \* *pIDs*, qapi\_Geofence\_Options\_t \* *options* )**

Modifies a specified number of Geofences.

##### Parameters

in	<i>count</i>	Number of Geofences to be modified.
in	<i>pIDs</i>	Array of IDs of the Geofences to be modified.
in	<i>options</i>	Array of structures containing the options: <ul style="list-style-type: none"> <li>• breachTypeMask – Bitwise OR of GeofenceBreachTypeMask bits</li> <li>• responsiveness in milliseconds</li> <li>• dwellTime in seconds</li> </ul>

##### Returns

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

#### 16.1.4.13 **qapi\_Location\_Error\_t qapi\_Loc\_Pause\_Geofences ( size\_t *count*, uint32\_t \* *pIDs* )**

Pauses a specified number of Geofences, which is similar to [qapi\\_Loc\\_Remove\\_Geofences\(\)](#) except that they can be resumed at any time.

##### Parameters

in	<i>count</i>	Number of Geofences to be paused.
in	<i>pIDs</i>	Array of IDs of the Geofences to be paused.

**Returns**

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

**16.1.4.14 qapi\_Location\_Error\_t qapi\_Loc\_Resume\_Geofences ( size\_t *count*,  
uint32\_t \* *pIDs* )**

Resumes a specified number of Geofences that are paused.

**Parameters**

in	<i>count</i>	Number of Geofences to be resumed.
in	<i>pIDs</i>	Array of IDs of the Geofences to be resumed.

**Returns**

- QAPI\_LOCATION\_ERROR\_SUCCESS if successful.
- QAPI\_LOCATION\_ERROR\_NOT\_INITIALIZED if no location session has been initialized.

# 17 Timer Module

---

This chapter describes the timer data types and APIs.

- [Timer APIs](#)
- [PMIC RTC APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 17.1 Timer APIs

This interface implements Advanced Time Services (ATS) timer services. This timer service is different than the RTOS timer service. This timer service will be available in SOM mode.

**Note:** These routines are fully re-entrant. In order to prevent memory leaks, whenever timer usage is done, the timer should be undefined using the `qapi_Timer_Undef()` API. Timer callbacks should do minimal processing. Time callbacks implementation should not contain any mutex or RPC.

\* Code snippet below demonstrates usage of timer interface. In the example  
 \* below, a client defines a timer, sets a timer, stops the timer, and  
 undefines a timer.  
 \* For Brevity, the sequence assumes that all calls succeed.

```
qapi_TIMER_handle_t timer_handle;

qapi_TIMER_def_attr_t timer_def_attr;
timer_def_attr.cb_type = TIMER_FUNC1_CB_TYPE; //notification type
timer_def_attr.sigs_func_ptr = &timer_test_cb; //callback to call when
//the timer expires
timer_def_attr.sigs_mask_data = 0x1; //this data will be returned in
//the callback
timer_def_attr.deferrable = false; //set to true for nondeferrable timer

//define the timer. Note: This call allocates memory and hence
//qapi_Timer_Undef() should be called whenever the timer usage is done.
qapi_Timer_def( &timer_handle, &timer_def_attr);

qapi_TIMER_set_attr_t timer_set_attr;
timer_set_attr.reload = FALSE; //Do not restart timer after it expires
timer_set_attr.time = time_duration;
timer_set_attr.unit = T_MSEC;

//set or start the timer
qapi_Timer_set( timer_handle, &timer_set_attr);

//stop a running timer
qapi_Timer_stop( timer_handle);

//Undef the timer. Releases memory allocated in qapi_Timer_Def()
qapi_Timer_undef( timer_handle);
```

### 17.1.1 Data Structure Documentation

#### 17.1.1.1 struct qapi\_TIMER\_define\_attr\_t

Timer define attribute type.

Type 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.
<a href="#">qapi_TIMER_notify_t</a>	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.

**17.1.1.2 struct qapi\_TIMER\_set\_attr\_t**

Timer set attribute type.

Type used to specify parameters when starting a timer.

**Data fields**

Type	Parameter	Description
uint64_t	time	Timer duration.
uint64_t	reload	Reload duration.
<a href="#">qapi_TIMER_unit_type</a>	unit	Specify units for timer duration.

**17.1.1.3 struct qapi\_TIMER\_get\_info\_attr\_t**

Timer information attribute type.

Type used to get information for a given timer.

**Data fields**

Type	Parameter	Description
<a href="#">qapi_TIMER_info_type</a>	timer_info	Timer information type.
<a href="#">qapi_TIMER_unit_type</a>	unit	Specify units to use for return.

**17.1.2 Typedef Documentation****17.1.2.1 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.

### 17.1.2.2 typedef void(\* qapi\_TIMER\_cb\_t)(uint32\_t data)

Timer callback type.

Timer callbacks should adhere to this signature.

## 17.1.3 Enumeration Type Documentation

### 17.1.3.1 enum qapi\_TIMER\_notify\_t

Timer notification type.

Enumeration of the notifications available on timer expiry.

**Enumerator:**

**QAPI\_TIMER\_NO\_NOTIFY\_TYPE** No notification.

**QAPI\_TIMER\_NATIVE\_OS\_SIGNAL\_TYPE** Signal an object.

**QAPI\_TIMER\_FUNC1\_CB\_TYPE** Call back a function.

### 17.1.3.2 enum qapi\_TIMER\_unit\_type

Timer unit type.

Enumeration of the units in which timer duration can be specified.

**Enumerator:**

**QAPI\_TIMER\_UNIT\_TICK** Return time in ticks.

**QAPI\_TIMER\_UNIT\_USEC** Return time in microseconds.

**QAPI\_TIMER\_UNIT\_MSEC** Return time in milliseconds.

**QAPI\_TIMER\_UNIT\_SEC** Return time in seconds.

**QAPI\_TIMER\_UNIT\_MIN** Return time in minutes.

**QAPI\_TIMER\_UNIT\_HOUR** Return time in hours.

### 17.1.3.3 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.



## 17.1.4 Function Documentation

### 17.1.4.1 **qapi\_Status\_t qapi\_Timer\_Def ( qapi\_TIMER\_handle\_t \* *timer\_handle*, qapi\_TIMER\_define\_attr\_t \* *timer\_attr* )**

Allocates internal memory in the timer module. The internal memory is then formatted with parameters provided in the *timer\_def\_attr* variable. The *timer\_handle* is returned to the client, and this handle must be used for any subsequent timer operations.

#### 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 will cause 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.

### 17.1.4.2 **qapi\_Status\_t qapi\_Timer\_Set ( qapi\_TIMER\_handle\_t *timer\_handle*, qapi\_TIMER\_set\_attr\_t \* *timer\_attr* )**

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.

#### 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.

### 17.1.4.3 **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* )**

Gets specified information about the timer.

**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.

#### 17.1.4.4 **qapi\_Status\_t qapi\_Timer\_Sleep ( uint64\_t *timeout*, qapi\_TIMER\_unit\_type *unit*, qbool\_t *non\_deferrable* )**

Timed wait. Blocks a thread for a specified time.

**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>	TRUE = processor (if in deep sleep or power collapse) will be awakened on timeout. FALSE = processor will not be awakened from deep sleep or power collapse on timeout. Whenever the processor wakes up due to some other reason after timeout, the thread will be unblocked.

**Returns**

QAPI\_OK on success, an error code on failure.

#### 17.1.4.5 **qapi\_Status\_t qapi\_Timer\_Undef ( qapi\_TIMER\_handle\_t *timer\_handle* )**

Undefines the timer. This API must be called whenever timer usage is done. Calling this API releases the internal timer memory that was allocated when the timer was defined.

**Parameters**

in	<i>timer_handle</i>	Timer handle for which to undefine the timer.
----	---------------------	---

**Returns**

QAPI\_OK on success, an error code on failure

#### 17.1.4.6 **qapi\_Status\_t qapi\_Timer\_Stop ( qapi\_TIMER\_handle\_t *timer\_handle* )**

Stops the timer.

**Note:** This function does not deallocate the memory that was allocated when the timer was defined.

**Parameters**

in	<i>timer_handle</i>	Timer handle for which to stop the timer.
----	---------------------	---

**Returns**

QAPI\_OK on success, an error code on failure.

**17.1.4.7 qapi\_Status\_t qapi\_Timer\_set\_absolute ( qapi\_TIMER\_handle\_t *timer*,  
uint64\_t *abs\_time* )**

Sets the timer with an expiry specified in absolute ticks.

**Parameters**

in	<i>timer</i>	Timer handle.
in	<i>abs_time</i>	Time tick when the timer expires.

**Returns**

QAPI\_OK on success, an error code on failure.

## 17.2 PMIC RTC APIs

This module provides the definitions to configure the real-time clock (RTC) alarm peripheral in the power management IC (PMIC).

### 17.2.1 Data Structure Documentation

#### 17.2.1.1 struct qapi\_PM\_Rtc\_Julian\_Type\_s

PMIC's version of the Julian time structure.

##### Data fields

Type	Parameter	Description
uint64_t	year	Year [1980 to 2100].
uint64_t	month	Month of the year [1 to 12].
uint64_t	day	Day of the month [1 to 31].
uint64_t	hour	Hour of the day [0 to 23].
uint64_t	minute	Minute of the hour [0 to 59].
uint64_t	second	Second of the minute [0 to 59].
uint64_t	day_of_week	Day of the week [0 to 6]; Monday through Sunday.

### 17.2.2 Typedef Documentation

#### 17.2.2.1 typedef struct qapi\_PM\_Rtc\_Julian\_Type\_s qapi\_PM\_Rtc\_Julian\_Type\_t

PMIC's version of the Julian time structure.

### 17.2.3 Enumeration Type Documentation

#### 17.2.3.1 enum qapi\_PM\_Rtc\_Cmd\_Type\_t

Real-time clock command type.

##### Enumerator:

**QAPI\_PM\_RTC\_SET\_CMD\_E** Set command.  
**QAPI\_PM\_RTC\_GET\_CMD\_E** Get command.

#### 17.2.3.2 enum qapi\_PM\_Rtc\_Display\_Type\_t

Real-time clock display mode type.

##### Enumerator:

**QAPI\_PM\_RTC\_12HR\_MODE\_E** 12 hour display mode.  
**QAPI\_PM\_RTC\_24HR\_MODE\_E** 24 hour display mode.

### 17.2.3.3 enum qapi\_PM\_Rtc\_Alarm\_Type\_t

RTC alarms.

**Enumerator:**

**QAPI\_PM\_RTC\_ALARM\_1\_E** Alarm 1.

**QAPI\_PM\_RTC\_ALL\_ALARMS\_E** Refers collectively to all supported alarms.

## 17.2.4 Function Documentation

### 17.2.4.1 qapi\_Status\_t qapi\_PM\_Rtc\_Init ( void )

Initializes the RTC after a power reset.

**Returns**

Possible values (see [qapi\\_Status\\_t](#)):

- QAPI\_OK – Operation succeeded.
- QAPI\_ERR\_NOT\_SUPPORTED – Feature is not supported.
- QAPI\_ERROR – Any other errors.

### 17.2.4.2 qapi\_Status\_t qapi\_PM\_Set\_Rtc\_Display\_Mode ( qapi\_PM\_Rtc\_Display\_Type\_t mode )

Configures the real time clock display mode (24 or 12 hour mode). The RTC defaults to 24 hr mode on phone power up and remains so until it is set to 12 hr mode explicitly using [qapi\\_PM\\_Set\\_Rtc\\_Display\\_Mode\(\)](#).

**Parameters**

in	mode	New RTC time display mode to be used. Valid values (see <a href="#">qapi_PM_Rtc_Display_Type_t</a> ): <ul style="list-style-type: none"> <li>• QAPI_PM_RTC_12HR_MODE_E</li> <li>• QAPI_PM_RTC_24HR_MODE_E</li> </ul>
----	------	--

**Returns**

Possible values (see [qapi\\_Status\\_t](#)):

- QAPI\_OK – Operation succeeded.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameter.
- QAPI\_ERR\_NOT\_SUPPORTED – Feature is not supported.
- QAPI\_ERROR – Any other errors.

### 17.2.4.3 **qapi\_Status\_t qapi\_PM\_Rtc\_Read\_Cmd ( qapi\_PM\_Rtc\_Julian\_Type\_t \* *qapi\_current\_time\_ptr* )**

Reads/writes the time and date from/to the PMIC RTC. The time/date format must be in 24 or 12 hr mode depending on in which mode the RTC was initialized. See the description of `qpai_PM_Set_Rtc_Display_Mode()` for details.

24 hr and 12 hr mode displays are:

24 HR – 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23

12 HR – 12 01 02 03 04 05 06 07 08 09 10 11 32 21 22 23 24 25 26 27 28 29 30 31

#### Parameters

in	<i>qapi_current_time_ptr</i>	Depending on the command, this function will use the <a href="#">qapi_PM_Rtc_Julian_Type_t</a> pointer to update or return the current time in the RTC.
----	------------------------------	---

#### Note

`day_of_week` is not required for setting the current time, but it returns the correct information when retrieving time from the RTC.

#### Returns

Possible values (see [qapi\\_Status\\_t](#)):

- QAPI\_OK – Operation succeeded.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameter.
- QAPI\_ERROR – Any other errors.

### 17.2.4.4 **qapi\_Status\_t qapi\_PM\_Rtc\_Alarm\_RW\_Cmd ( qapi\_PM\_Rtc\_Cmd\_Type\_t *cmd*, qapi\_PM\_Rtc\_Alarm\_Type\_t *what\_alarm*, qapi\_PM\_Rtc\_Julian\_Type\_t \* *qapi\_alarm\_time\_ptr* )**

Reads/writes the time and date from/to the PMIC RTC. The time/date format must be in 24 or 12 hr mode depending on in which mode the RTC was initialized. See the description of `qpai_PM_Set_Rtc_Display_Mode()` for details.

24 hr and 12 hr mode displays are:

24 HR – 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23

12 HR – 12 01 02 03 04 05 06 07 08 09 10 11 32 21 22 23 24 25 26 27 28 29 30 31

#### Parameters

in	<i>cmd</i>	Indicates whether to set or get the current time in the RTC. Valid values (see <code>qapi_PM_Rtc_Cmd_Type_t</code> ): <ul style="list-style-type: none"> <li>• QAPI_PM_RTC_SET_CMD_E</li> <li>• QAPI_PM_RTC_GET_CMD_E</li> </ul>
in	<i>what_alarm</i>	Alarm type. See <a href="#">qapi_PM_Rtc_Alarm_Type_t</a> .

in	<i>qapi_alarm_time_ptr</i>	Depending on the command, this function will use the structure <a href="#">qapi_PM_Rtc_Julian_Type_t</a> pointer to update or return the alarm time in the RTC.
----	----------------------------	---

**Note**

day\_of\_week is not required for setting the current time, but it returns the correct information when retrieving time from the RTC.

**Returns**

Possible values (see [qapi\\_Status\\_t](#)):

- QAPI\_OK – Operation succeeded.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameter.
- QAPI\_ERROR – Any other errors.

# 18 Hardware Engine APIs

---

This chapter describes the ADC and TSENS data types and APIs.

- [ADC Data Types](#)
- [ADC APIs](#)
- [TSENS Data Types](#)
- [TSENS APIs](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com



## 18.1 ADC Data Types

### 18.1.1 Define Documentation

#### 18.1.1.1 #define ADC\_INPUT\_BATT\_ID "BATT\_ID"

Physical units are in millivolts.

#### 18.1.1.2 #define ADC\_INPUT\_PA\_THERM "PA\_THERM"

Physical units are in degrees C.

#### 18.1.1.3 #define ADC\_INPUT\_PA\_THERM1 "PA\_THERM1"

Physical units are in degrees C.

#### 18.1.1.4 #define ADC\_INPUT\_PMIC\_THERM "PMIC\_THERM"

Physical units are in 0.001 gradients of degrees C.

#### 18.1.1.5 #define ADC\_INPUT\_VBATT "VBATT"

Physical units are in millivolts.

#### 18.1.1.6 #define ADC\_INPUT\_VPH\_PWR "VPH\_PWR"

Physical units are in millivolts.

#### 18.1.1.7 #define ADC\_INPUT\_XO\_THERM "XO\_THERM"

Physical units are in  $2^{-10}$  degrees C.

#### 18.1.1.8 #define ADC\_INPUT\_XO\_THERM\_GPS "XO\_THERM\_GPS"

Physical units are in  $2^{-10}$  degrees C.

### 18.1.2 Data Structure Documentation

#### 18.1.2.1 struct qapi\_ADC\_Read\_Result\_t

ADC read results.

##### Data fields

Type	Parameter	Description
unsigned int	eStatus	Status of the conversion.
uint32_t	nToken	Token that identifies the conversion.
uint32_t	nDeviceIdx	Device index for the conversion.
uint32_t	nChannelIdx	Channel index for the conversion.
int32_t	nPhysical	Result in physical units. Units depends on the BSP.

Type	Parameter	Description
uint32_t	nPercent	Result as a percentage of the reference voltage used for the conversion: 0 = 0%, 65535 = 100%
uint32_t	nMicrovolts	Result in microvolts.
uint32_t	nCode	Raw ADC code from the hardware.

### 18.1.2.2 struct qapi\_Adc\_Input\_Properties\_Type\_t

ADC input properties.

#### Data fields

Type	Parameter	Description
uint32_t	nDeviceIdx	Device index.
uint32_t	nChannelIdx	Channel index.

### 18.1.2.3 struct qapi\_AdcTM\_Input\_Properties\_Type\_t

ADC TM input properties.

#### Data fields

Type	Parameter	Description
uint32_t	nDeviceIdx	Device index.
uint32_t	nChannelIdx	Channel index.

### 18.1.2.4 struct qapi\_ADC\_Range\_t

ADC range structure.

#### Data fields

Type	Parameter	Description
int32_t	min_uv	Minimum value in microvolts.
int32_t	max_uv	Maximum value in microvolts.

### 18.1.2.5 struct qapi\_ADC\_Threshold\_Result\_t

ADC amplitude threshold result structure.

#### Data fields

Type	Parameter	Description
uint32_t	channel	Channel that was triggered.
qapi_AD-C_Amp-Threshold_t	threshold	Threshold that was triggered.

**18.1.2.6 struct qapi\_ADC\_Device\_Properties\_t**

ADC device properties structure.

**Data fields**

Type	Parameter	Description
uint32_t	uNumChannels	Number of ADC channels.

**18.1.2.7 struct qapi\_AdcTM\_Callback\_Payload\_Type\_t**

ADC TM callback payload structure

**Data fields**

Type	Parameter	Description
qapi_AD-C_Amp_Threshold_t	eThreshold-Triggered	Type of threshold that triggered.
uint32_t	uTMChannel-Idx	TM channel index.
int32_t	nPhysical-Triggered	Physical value that triggered.

**18.1.2.8 struct qapi\_AdcTM\_Range\_Type\_t**

ADC TM channel range structure.

**Data fields**

Type	Parameter	Description
int32_t	nPhysicalMin	Minimum threshold in physical units.
int32_t	nPhysicalMax	Maximum threshold in physical units.

**18.1.2.9 struct qapi\_AdcTM\_Request\_Params\_Type\_t**

ADC TM request parameters structure.

**Data fields**

Type	Parameter	Description
qapi_Adc-_Input_Properties_Type_t	adcTMInput-Props	ADC channel input properties.
qapi_AdcTM_Threshold_Cb_Type	pfnAdcTM-ThresholdCb	Amplitude threshold callback type.
void *	pCtxt	Context specified when setting the threshold.

## 18.1.3 Typedef Documentation

### 18.1.3.1 typedef void(\* qapi\_ADC\_Threshold\_CB\_t)(void \*ctxt, const qapi\_ADC\_Threshold\_Result\_t \*result)

Callback invoked when an amplitude threshold is crossed.

Once the threshold is crossed, it must be re-armed or it will not trigger again.

#### Parameters

in	<i>ctxt</i>	Context specified when setting the threshold.
in	<i>result</i>	Threshold crossing result.

#### Returns

None.

### 18.1.3.2 typedef void(\* qapi\_AdcTM\_Threshold\_Cb\_Type)(void \*ctxt, const qapi\_ADC\_Threshold\_Result\_t \*result)

Callback invoked when an amplitude threshold is crossed.

Once the threshold is crossed, it must be re-armed or it will not trigger again.

#### Parameters

in	<i>ctxt</i>	Context specified when setting the threshold.
in	<i>result</i>	Threshold crossing result.

#### Returns

None.

## 18.1.4 Enumeration Type Documentation

### 18.1.4.1 enum qapi\_ADC\_Amp\_Threshold\_t

ADC amplitude threshold types that can be configured to be monitored using qapi\_ADC\_Set\_Threshold().

#### Enumerator:

**QAPI\_ADC\_THRESHOLD\_LOWER\_E** Lower threshold.  
**QAPI\_ADC\_THRESHOLD\_HIGHER\_E** Higher threshold.

## 18.2 ADC APIs

The analog-to-digital converter (ADC) allows an analog signal to be sampled and digitally represented. The SoC features an on-die ADC that supports reading multiple channels. The ADC can perform single-shot and recurring measurements.

The ADC is configurable via static parameters. See the ADC tunable board file for the statically defined parameters.

This programming interface allows client software to configure channels, perform single readings, set a threshold if the channel is an ADC TM channel before reading the channel, and get ADC data samples. The code snippet below shows an example usage.

\* The code snippet below demonstrates use of this interface. The example  
 \* below opens ADC to obtain a handle, sets the thresholds if the channel  
 \* is an ADC TM channel, reads each ADC channel, and then closes the handle.

```
qapi_Status_t status;
qapi_ADC_Handle_t handle;
uint32_t num_channels;
uint32_t channel;
qapi_ADC_Read_Result_t result;
const char Channel_Name;
uint32_t Channel_Name_Size;
qapi_AdcTM_Input_Properties_Type_t Properties_TM;
qapi_Adc_Input_Properties_Type_t Properties;
uint32_t Enable;
const qapi_AdcTM_Request_Params_Type_t ADC_TM_Params, TM_Params_Type;
const int32 Lower_Tolerance, Higher_Tolerance, Threshold_Desired;
qapi_ADC_Amp_Threshold_t Threshold_Type;
qapi_AdcTM_Range_Type_t ADC_TM_Range;
int32 TM_Threshold_Set;

status = qapi_ADC_Open(&handle, Dummy);
if (status != QAPI_OK) { ... }

//To read ADC channels
status=qapi_ADC_Get_Input_Properties(&handle, Channel_Name,
                                   Channel_Name_Size, Properties);
if (status != QAPI_OK) { ... }

// To read and configure ADC TM channels
status=qapi_ADC_TM_Get_Input_Properties(&handle, Channel_Name,
                                       Channel_Name_Size, Properties_TM);
if (status != QAPI_OK) { ... }
else
{
    status=qapi_ADC_Get_Range(&handle, channel, ADC_TM_Range);
    if (status != QAPI_OK) { ... }

    status=qapi_ADC_Set_Amp_Threshold(&handle, ADC_TM_Params,
                                     Threshold_Type, Threshold_Desired, TM_Threshold_Set);
    if (status != QAPI_OK) { ... }

    //Enable Thresholds (Enable = 1)
    status=qapi_ADC_TM_Enable_Thresholds(&handle, Enable, Threshold_Type);
    if (status != QAPI_OK) { ... }

    status=qapi_ADC_TM_Set_Tolerance(&handle, TM_Params_Type_Ptr,
                                    Lower_Tolerance, Higher_Tolerance);
```

```

if (status != QAPI_OK) { ... }

//Disable Thresholds (Enable = 0)
status=qapi_ADC_TM_Enable_Thresholds(&handle, Enable, Threshold_Type);
if (status != QAPI_OK) { ... }
}

for (channel = 0; channel < num_channels; channel++)
{
    status = qapi_ADC_Read_Channel(handle, channel, &result);
    if (status != QAPI_OK) { ... }

    // result.microvolts contains the reading
}
status = qapi_ADC_Close(handle, false);
if (status != QAPI_OK) { ... }
handle = NULL;

```

## 18.2.1 Function Documentation

### 18.2.1.1 **qapi\_Status\_t qapi\_ADC\_Open ( qapi\_ADC\_Handle\_t \* *Handle*, uint32\_t *Attributes* )**

Opens the ADC for use by a software client.

ADC clients values can only be read after successfully opening ADC.

#### Parameters

out	<i>Handle</i>	Pointer to an ADC handle.
in	<i>Attributes</i>	Reserved parameter.

#### Returns

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.
- QAPI\_ERR\_NO\_MEMORY – No memory available to support this operation.
- QAPI\_ERR\_NO\_RESOURCE – No more handles are available.

### 18.2.1.2 **qapi\_Status\_t qapi\_ADC\_Get\_Input\_Properties ( qapi\_ADC\_Handle\_t *Handle*, const char \* *Channel\_Name\_Ptr*, uint32\_t *Channel\_Name\_Size*, qapi\_Adc\_Input\_Properties\_Type\_t \* *Properties\_Ptr* )**

Gets the ADC channel configuration.

This function is used to get properties of ADC channels.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>Channel_Name_Ptr</i>	Pointer to ADC channel name pointer.
in	<i>Channel_Name_Size</i>	Size of channel name string.
out	<i>Properties_Ptr</i>	ADC channel configuration.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.2.1.3 **qapi\_Status\_t qapi\_ADC\_Read\_Channel ( qapi\_ADC\_Handle\_t *Handle*, const qapi\_Adc\_Input\_Properties\_Type\_t \* *Input\_Prop\_Ptr*, qapi\_ADC\_Read\_Result\_t \* *Result\_Ptr* )**

Reads an ADC channel.

This function performs a blocking ADC read for the device and channel specified by the client in pAdcInputProps.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>Input_Prop_Ptr</i>	Properties pointer of channel provided by <a href="#">qapi_ADC_Get_Input_Properties()</a> .
out	<i>Result_Ptr</i>	ADC reading result structure.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.2.1.4 **qapi\_Status\_t qapi\_ADC\_TM\_Get\_Input\_Properties ( qapi\_ADC\_Handle\_t *Handle*, const char \* *Channel\_Name\_Ptr*, uint32\_t *Channel\_Name\_Size*, qapi\_AdcTM\_Input\_Properties\_Type\_t \* *Properties\_Ptr* )**

Gets the ADC TM channel configuration.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>Channel_Name_Ptr</i>	Pointer to the ADC TM channel name pointer.
in	<i>Channel_Name_Size</i>	Size of channel name string.
out	<i>Properties_Ptr</i>	ADC TM channel configuration.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.2.1.5 **qapi\_Status\_t qapi\_ADC\_Get\_Range ( qapi\_ADC\_Handle\_t *Handle*, const qapi\_AdcTM\_Input\_Properties\_Type\_t \* *In\_Properties\_Ptr*, qapi\_AdcTM\_Range\_Type\_t \* *ADC\_TM\_Range\_Ptr* )**

Gets the ADC TM channels range of operation.

This function gets the minimum and maximum physical value that can be set as a threshold for a given VADC TM channel.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>In_Properties_Ptr</i>	Properties pointer of the channel provided by <a href="#">qapi_ADC_TM_Get_Input_Properties()</a> .
out	<i>ADC_TM_Range_Ptr</i>	Pointer to the channel range.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.2.1.6 **qapi\_Status\_t qapi\_ADC\_Set\_Amp\_Threshold ( qapi\_ADC\_Handle\_t *Handle*, const qapi\_AdcTM\_Request\_Params\_Type\_t \* *ADC\_TM\_Params\_Ptr*, qapi\_ADC\_Amp\_Threshold\_t *Threshold\_Type*, const int32 \* *Threshold\_Desired\_Ptr*, int32 \* *TM\_Threshold\_Set\_Ptr* )**

Sets the threshold-related configuration for ADC TM channels.

The threshold event is triggered once when the threshold is crossed:

- ADC\_TM\_THRESHOLD\_LOWER: current reading  $\leq$  \*Threshold\_Desired\_Ptr
- ADC\_TM\_THRESHOLD\_HIGHER: current reading  $\geq$  \*Threshold\_Desired\_Ptr

After the event is triggered, the threshold will not trigger the event again and will be in a triggered state until the client calls [qapi\\_ADC\\_Set\\_Amp\\_Threshold\(\)](#) to set a new threshold.

Note that thresholds can be disabled/re-enabled on a per client basis by calling [qapi\\_ADC\\_Clear\\_Amp\\_Threshold\(\)](#). Thresholds are enabled by default, but calling [qapi\\_ADC\\_Clear\\_Amp\\_Threshold\(\)](#) does not automatically re-enable them if they were previously disabled by a call to [qapi\\_ADC\\_Clear\\_Amp\\_Threshold\(\)](#).



**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>ADC_TM_Params_Ptr</i>	Pointer to the threshold parameters.
in	<i>Threshold_Type</i>	Type of threshold.
in	<i>Threshold_Desired_Ptr</i>	Pointer to desired threshold value.
out	<i>TM_Threshold_Set_Ptr</i>	Pointer to threshold value actually set.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.2.1.7 **qapi\_Status\_t qapi\_ADC\_TM\_Enable\_Thresholds ( qapi\_ADC\_Handle\_t *Handle*, uint32\_t *Enable*, qapi\_ADC\_Amp\_Threshold\_t *Threshold\_Type* )**

Enables or Disables thresholds on ADC TM channel. By default, thresholds are enabled.

Thresholds are not monitored while they are disabled, and any threshold crossings that occurred while the thresholds were disabled are ignored.

Threshold values and event handles set by [qapi\\_ADC\\_Set\\_Amp\\_Threshold\(\)](#) are retained while thresholds are disabled.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>Enable</i>	Enable or disable thresholds.
in	<i>Threshold_Type</i>	Type of threshold.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.2.1.8 **qapi\_Status\_t qapi\_ADC\_TM\_Set\_Tolerance ( qapi\_ADC\_Handle\_t *Handle*, const qapi\_AdcTM\_Request\_Params\_Type\_t \* *TM\_Params\_Type*, const int32 \* *Lower\_Tolerance*, const int32 \* *Higher\_Tolerance* )**

Sets thresholds based on an allowable tolerance or delta.

This API allows clients to specify a tolerance for how much the measurement can change before being notified, e.g., notify when XO\_THERM changes by 0.02 degrees C. Thresholds are set based on the current

measurement value +/- the allowable delta.

Once the tolerance has been reached or exceeded, the ADC notifies the client and automatically sets new thresholds for the tolerance. Clients must clear the tolerances for the ADC to stop monitoring. Tolerances can be cleared by setting a NULL value.

Clients can set or clear either a low tolerance, high tolerance, or both during the same function call. If the client is already monitoring a tolerance, setting a new tolerance results in an update to the previously set tolerance, i.e., the new tolerance replaces the old tolerance.

A client can set either a threshold or a tolerance on any one measurement, but not both at the same time. To allow a threshold to be set after registering a tolerance, the tolerance must be cleared by passing in NULL parameters for the tolerances.

The client event is triggered when the tolerance is met or exceeded:

- Lower: The event triggers when the `current_value`  $\leq$  `original_value` - tolerance
- Upper: The event triggers when the `current_value`  $\geq$  `original_value` + tolerance

#### Parameters

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>TM_Params_Type</i>	Pointer to threshold configuration of ADCM TM channel.
in	<i>Lower_Tolerance</i>	Pointer to lower tolerance.
in	<i>Higher_Tolerance</i>	Pointer to higher tolerance.

#### Returns

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

#### 18.2.1.9 `qapi_Status_t qapi_ADC_Close ( qapi_ADC_Handle_t Handle, qbool_t keep_enabled )`

Closes a handle to the ADC when a software client is done with it.

#### Parameters

in	<i>Handle</i>	Handle provided by <a href="#">qapi_ADC_Open()</a> .
in	<i>keep_enabled</i>	Reserved parameter.

#### Returns

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

## 18.3 TSENS Data Types

This section provides the type definitions for temperature sensor APIs.

### 18.3.1 Data Structure Documentation

#### 18.3.1.1 struct qapi\_TSENS\_CallbackPayloadType\_t

TSENS callback payload type structure.

##### Data fields

Type	Parameter	Description
qapi_TSENS_ThresholdType_t	eThreshold	Type of threshold that was triggered.
unsigned int	uSensor	Sensor that was triggered.
int	nTriggeredDeg-C	Temperature value that was triggered.

#### 18.3.1.2 struct qapi\_TSENS\_Result\_t

TSENS temperature result structure.

##### Data fields

Type	Parameter	Description
int	deg_c	Temperature in degrees Celsius.

## 18.3.2 Typedef Documentation

**18.3.2.1** `typedef void(* QAPI_Tsens_Threshold_Cb_Type)(void *pCtxt, const qapi_TSENS_CallbackPayloadType_t *pPayload)`

TSENS callback function type.

**18.3.2.2** `typedef void* qapi_TSENS_Handle_t`

TSENS handler type.

## 18.3.3 Enumeration Type Documentation

**18.3.3.1** `enum qapi_TSENS_ThresholdType_t`

Enumeration of TSENS temperature thresholds.

**Enumerator:**

**QAPIS\_TSENS\_THRESHOLD\_LOWER** Lower threshold.

**QAPIS\_TSENS\_THRESHOLD\_UPPER** Upper threshold.

**QAPIS\_TSENS\_NUM\_THRESHOLDS** Number of thresholds.

## 18.4 TSENS APIs

The temperature sensor is used to monitor the temperature of the SoC using on-die analog sensors.

This programming interface allows client software to read the temperature returned by each sensor. The code snippet below shows an example usage.

Consult hardware documentation for the placement of the sensors on the die.

```
* The code snippet below demonstates usage of this interface. The example
* below opens TSENS to obtain a handle, gets the number of sensors, sets
* temparature thresholds for each sensor, reads each sensor's
* temperature, and then closes the handle.
```

```
qapi_Status_t status;
qapi_TSENS_Handle_t handle;
uint32_t num_sensors;
uint32_t sensor;
qapi_TSENS_Result_t result;
qapi_TSENS_ThresholdType_t Threshold_Type;
int32_t Threshold_Degree;
QAPI_Tsens_Threshold_Cb_Type Threshold_CB;

status = qapi_TSENS_Open(&handle);
if (status != QAPI_OK) { ... }

status = qapi_TSENS_Get_Num_Sensors(handle, &num_sensors);
if (status != QAPI_OK) { ... }

for (sensor = 0; sensor < num_sensors; sensor++)
{
    status = qapi_TSENS_Get_Calibration_Status(handle, sensor, &result);
    if (status != QAPI_OK) { ... }

    else
    {
        status=qapi_TSENS_Get_Temp(handle, sensor, &result);
        if (status != QAPI_OK) { ... }

        else
        {
            status= qapi_TSENS_Set_Thresholds(handle, sensor,
                Threshold_Type, Threshold_Degree,
                Threshold_CB, context_ptr);
            if (status != QAPI_OK) { ... }

            else
            {
                status=qapi_TSENS_Set_Enable_Thresholds(handle,enable);
                if (status != QAPI_OK) { ... }
            }
        }
    }

    // result->Deg_C is the temperature in degrees Celsius
}

status = qapi_TSENS_Close(handle);
if (status != QAPI_OK) { ... }
handle = NULL;
```

## 18.4.1 Function Documentation

### 18.4.1.1 `qapi_Status_t qapi_TSENS_Open ( qapi_TSENS_Handle_t * Handle )`

Opens TSENS.

#### Parameters

out	<i>Handle</i>	Pointer to a TSENS handle.
-----	---------------	----------------------------

#### Returns

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.4.1.2 `qapi_Status_t qapi_TSENS_Get_Num_Sensors ( qapi_TSENS_Handle_t Handle, int32_t * Num_Sensors_Ptr )`

Gets the number of TSENS sensors.

This function gets the number of TSENS sensors supported by the SoC. The sensor index is zero-based and ranges from 0 to the number of sensors minus one.

#### Parameters

in	<i>Handle</i>	Handle provided by <a href="#">qapi_TSENS_Open()</a> .
out	<i>Num_Sensors_Ptr</i>	Number of sensors

#### Returns

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

### 18.4.1.3 `qapi_Status_t qapi_TSENS_Get_Temp ( qapi_TSENS_Handle_t Handle, uint32_t Sensor_Num, qapi_TSENS_Result_t * Temp_Result_Ptr )`

Gets the temperature of a specified sensor.

This function waits until a measurement is complete. This means the calling thread can be blocked by up to several hundredths of microseconds. The exact delay depends on the number of sensors present in the hardware and the hardware conversion time per sensor. There is a fixed timeout value built into this function. If the measurement does not complete before the timeout, this function returns TSENS\_ERROR\_TIMEOUT.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_TSENS_Open()</a> .
in	<i>Sensor_Num</i>	Selected sensor
out	<i>Temp_Result_Ptr</i>	Temperature reported by the sensor.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

#### 18.4.1.4 **qapi\_Status\_t qapi\_TSENS\_Get\_Calibration\_Status ( qapi\_TSENS\_Handle\_t *Handle*, uint32\_t *Sensor\_Num* )**

Gets the calibration status for a temperature sensor.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_TSENS_Open()</a> .
in	<i>Sensor_Num</i>	Selected sensor number.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.
- QAPI\_ERR\_TIMEOUT – The sensor did not return a reading before the timeout.

#### 18.4.1.5 **qapi\_Status\_t qapi\_TSENS\_Set\_Thresholds ( qapi\_TSENS\_Handle\_t *Handle*, uint32\_t *Sensor\_Num*, qapi\_TSENS\_ThresholdType\_t *Threshold\_Type*, int32\_t *Threshold\_Degree*, QAPI\_Tsens\_Threshold\_Cb\_Type *Threshold\_CB*, void \* *Context\_Ptr* )**

Sets the threshold for a sensor.

The threshold event is triggered once when the threshold is crossed. After the event is triggered, the threshold will not trigger the event again and will be in a triggered state until the client calls this function again to set a new threshold.

Note that thresholds can be disabled/reenabled on a per client basis by calling [qapi\\_TSENS\\_Set\\_Enable\\_Thresholds\(\)](#). Thresholds are enabled by default, but calling [qapi\\_TSENS\\_Set\\_Thresholds\(\)](#) does not automatically reenables them if they were previously disabled by a call to [qapi\\_TSENS\\_Set\\_Enable\\_Thresholds\(\)](#).

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_TSENS_Open()</a> .
in	<i>Sensor_Num</i>	Selected sensor.
in	<i>Threshold_Type</i>	Threshold typeSelected sensor.
in	<i>Threshold_Degree</i>	Threshold in degrees centigrade.
in	<i>Threshold_CB</i>	Threshold callback.
in	<i>Context_Ptr</i>	Context pointer that is returned with the callback.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

#### 18.4.1.6 **qapi\_Status\_t qapi\_TSENS\_Set\_Enable\_Thresholds ( qapi\_TSENS\_Handle\_t *Handle*, int32\_t *Enable\_Threshold* )**

Sets enable/disable of a specified sensor.

Enables or disables the upper and lower thresholds that were registered by this client by calls to [qapi\\_TSENS\\_Set\\_Thresholds\(\)](#). By default, thresholds are enabled.

Thresholds are not monitored while the thresholds are disabled, and any threshold crossings that occurred while the thresholds were disabled are ignored.

Threshold values and event handles set by `DalTsens_SetThreshold` are still retained while thresholds are disabled. This does not affect the critical thresholds. Critical thresholds are always enabled.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_TSENS_Open()</a> .
in	<i>Enable_Threshold</i>	Enable or disable the threshold.

**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

#### 18.4.1.7 **qapi\_Status\_t qapi\_TSENS\_Close ( qapi\_TSENS\_Handle\_t *Handle* )**

Closes TSENS.

**Parameters**

in	<i>Handle</i>	Handle provided by <a href="#">qapi_TSENS_Open()</a> .
----	---------------	--



**Returns**

- QAPI\_OK – Call succeeded.
- QAPI\_ERROR – Call failed.
- QAPI\_ERR\_INVALID\_PARAM – Invalid parameters were specified.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

# 19 System Power Save Management

---

This chapter describes the system power save management (PSM) data types and APIs.

- [PSM Data Types](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

## 19.1 PSM Data Types

This section provides type definitions for PSM client APIs.

### PSM Client Status Messages

- #define [QAPI\\_ERR\\_PSM\\_FAIL](#) \_\_QAPI\_PSM\_ERROR(1)
- #define [QAPI\\_ERR\\_PSM\\_GENERIC\\_FAILURE](#) \_\_QAPI\_PSM\_ERROR(2)
- #define [QAPI\\_ERR\\_PSM\\_APP\\_NOT\\_REGISTERED](#) \_\_QAPI\_PSM\_ERROR(3)
- #define [QAPI\\_ERR\\_PSM\\_WRONG\\_ARGUMENTS](#) \_\_QAPI\_PSM\_ERROR(4)
- #define [QAPI\\_ERR\\_PSM\\_IPC\\_FAILURE](#) \_\_QAPI\_PSM\_ERROR(5)

### 19.1.1 Define Documentation

#### 19.1.1.1 #define QAPI\_ERR\_PSM\_FAIL \_\_QAPI\_PSM\_ERROR(1)

PSM failed.

#### 19.1.1.2 #define QAPI\_ERR\_PSM\_GENERIC\_FAILURE \_\_QAPI\_PSM\_ERROR(2)

Generic failure.

#### 19.1.1.3 #define QAPI\_ERR\_PSM\_APP\_NOT\_REGISTERED \_\_QAPI\_PSM\_ERROR(3)

Application is not registred.

#### 19.1.1.4 #define QAPI\_ERR\_PSM\_WRONG\_ARGUMENTS \_\_QAPI\_PSM\_ERROR(4)

Wrong arguments.

#### 19.1.1.5 #define QAPI\_ERR\_PSM\_IPC\_FAILURE \_\_QAPI\_PSM\_ERROR(5)

IPC failure.

### 19.1.2 Data Structure Documentation

#### 19.1.2.1 struct psm\_time\_info\_type

PSM time information.

##### Data fields

Type	Parameter	Description
<a href="#">psm_time_-format_type_e</a>	time_format_-flag	Time format. See <a href="#">psm_time_format_type_e</a> .
pm_rtc_julian_-type	wakeup_time	Time in broken down format if time_format_flag is set to PSM_TIME_IN_TM.
int	psm_duration_-in_secs	Time in seconds if time_format_flag is set to PSM_TIME_IN_SECS.

### 19.1.2.2 struct psm\_info\_type

PSM information type.

#### Data fields

Type	Parameter	Description
int	active_time_in_secs	Active time is the duration the PSM server must wait before entering PSM mode. The purpose of this time is to provide a chance for the MTC server to react.
<a href="#">psm_wakeup_type_e</a>	psm_wakeup_type	Next wakeup from PSM mode is for measurement purpose or measurement and network access.
<a href="#">psm_time_info_type</a>	psm_time_info	PSM time information. See <a href="#">psm_time_info_type</a> .

### 19.1.2.3 struct psm\_status\_msg\_type

PSM status message type.

#### Data fields

Type	Parameter	Description
int	client_id	Client ID.
int	status	PSM status. See <a href="#">psm_status_type_e</a> .
int	reason	PSM reject reason. See <a href="#">psm_reject_reason_type_e</a> .

## 19.1.3 Typedef Documentation

### 19.1.3.1 typedef void(\* psm\_client\_cb\_type)(psm\_status\_msg\_type \*)

PSM status callback type.

### 19.1.3.2 typedef void(\* psm\_util\_timer\_expiry\_cb\_type)(void \*, size\_t)

PSM timer expiry callback type.

## 19.1.4 Enumeration Type Documentation

### 19.1.4.1 enum psm\_status\_type\_e

Enumeration of status types.

#### Enumerator:

**PSM\_STATUS\_REJECT** PSM enter request is rejected.

**PSM\_STATUS\_READY** Ready to enter PSM mode.

**PSM\_STATUS\_NOT\_READY** Not ready to enter PSM.

**PSM\_STATUS\_COMPLETE** Entered PSM mode, the system might shutdown at any time.

**PSM\_STATUS\_DISCONNECTED** PSM server is down.

**PSM\_STATUS\_MODEM\_LOADED** Modem is loaded as part of bootup.

**PSM\_STATUS\_MODEM\_NOT\_LOADED** Modem is not loaded as part of bootup.

**PSM\_STATUS\_NW\_OOS** Network is OOS.

**PSM\_STATUS\_NW\_LIMITED\_SERVICE** Network is in limited service.

**PSM\_STATUS\_HEALTH\_CHECK** Application health check.

**PSM\_STATUS\_FEATURE\_ENABLED** Feature is dynamically enabled.

**PSM\_STATUS\_FEATURE\_DISABLED** Feature is dynamically disabled.

#### 19.1.4.2 enum psm\_reject\_reason\_type\_e

Enumeration of reasons for rejection.

Enumerator:

**PSM\_REJECT\_REASON\_NONE** No reject reason.

**PSM\_REJECT\_REASON\_NOT\_ENABLED** PSM feature is not enabled.

**PSM\_REJECT\_REASON\_MODEM\_NOT\_READY** Modem is not ready to enter PSM mode.

**PSM\_REJECT\_REASON\_DURATION\_TOO\_SHORT** PSM duration is too short to enter PSM mode.

#### 19.1.4.3 enum psm\_error\_type\_e

Enumeration of PSM error types.

Enumerator:

**PSM\_ERR\_NONE** Success.

**PSM\_ERR\_FAIL** Failure.

**PSM\_ERR\_GENERIC\_FAILURE** Miscellaneous failure.

**PSM\_ERR\_APP\_NOT\_REGISTERED** Application is not registered with the PSM server.

**PSM\_ERR\_WRONG\_ARGUMENTS** Wrong input arguments.

**PSM\_ERR\_IPC\_FAILURE** Failure to communicate with the PSM server.

#### 19.1.4.4 enum psm\_time\_format\_type\_e

PSM time format.

Enumerator:

**PSM\_TIME\_IN\_TM** Specify time in broken down format.

**PSM\_TIME\_IN\_SECS** Specify time in seconds.

#### 19.1.4.5 enum psm\_wakeup\_type\_e

PSM wakeup type.

Enumerator:

**PSM\_WAKEUP\_MEASUREMENT\_ONLY** Next wakeup from PSM is for measurement purpose only.

**PSM\_WAKEUP\_MEASUREMENT\_NW\_ACCESS** Next wakeup from PSM is for measurement and network access.

## 20 Device Information Module

---

This chapter describes the device information data types and APIs.

- [Device Information](#)

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qcomtel.com

## 20.1 Device Information

### 20.1.1 Define Documentation

#### 20.1.1.1 #define QAPI\_DEVICE\_INFO\_BUF\_SIZE 128

Maximum size of `qapi_Device_Info_t` valuebuf.

### 20.1.2 Data Structure Documentation

#### 20.1.2.1 struct qapi\_Device\_Info\_t

QAPI device information structure.

##### Data fields

Type	Parameter	Description
<code>qapi_Device_Info_ID_t</code>	id	Required information ID.
<code>qapi_Device_Info_Type_t</code>	info_type	Response type.
union <code>qapi_Device_Info_t</code>	u	

#### 20.1.2.2 union qapi\_Device\_Info\_t.u

##### Data fields

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

#### 20.1.2.3 struct qapi\_Device\_Info\_t.u.valuebuf

##### Data fields

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

## 20.1.3 Enumeration Type Documentation

### 20.1.3.1 enum qapi\_Device\_Info\_ID\_t

Device information types.

Enumerator:

**QAPI\_DEVICE\_INFO\_BUILD\_ID\_E** Device BUILD\_ID.  
**QAPI\_DEVICE\_INFO\_IMEI\_E** Device IMEI.  
**QAPI\_DEVICE\_INFO\_IMSI\_E** UIM IMSI.  
**QAPI\_DEVICE\_INFO\_OS\_VERSION\_E** Device OS version.  
**QAPI\_DEVICE\_INFO\_MANUFACTURER\_E** Device manufacturer.  
**QAPI\_DEVICE\_INFO\_MODEL\_ID\_E** Device model ID.  
**QAPI\_DEVICE\_INFO\_BATTERY\_STATUS\_E** Device battery status.  
**QAPI\_DEVICE\_INFO\_BATTERY\_PERCENTAGE\_E** Device battery percentage.  
**QAPI\_DEVICE\_INFO\_TIME\_ZONE\_E** Device time zone.  
**QAPI\_DEVICE\_INFO\_ICCID\_E** Device ICCID.  
**QAPI\_DEVICE\_INFO\_4G\_SIG\_STRENGTH\_E** Network signal strength.  
**QAPI\_DEVICE\_INFO\_BASE\_STATION\_ID\_E** Network base station ID.  
**QAPI\_DEVICE\_INFO\_MCC\_E** Network MCC.  
**QAPI\_DEVICE\_INFO\_MNC\_E** Network MNC.  
**QAPI\_DEVICE\_INFO\_SERVICE\_STATE\_E** Network service status.  
**QAPI\_DEVICE\_INFO\_MBN\_E** Device MBN.  
**QAPI\_DEVICE\_INFO\_TAC\_E** Network tracking area code.  
**QAPI\_DEVICE\_INFO\_CELL\_ID\_E** Network cell ID.  
**QAPI\_DEVICE\_INFO\_RCCS\_E** Network RRC state.  
**QAPI\_DEVICE\_INFO\_EMMS\_E** Network EMM state.  
**QAPI\_DEVICE\_INFO\_EMMSUBS\_E** Network EMM substate.  
**QAPI\_DEVICE\_INFO\_SERVING\_RSRQ\_E** Serving cell RSRQ.  
**QAPI\_DEVICE\_INFO\_NEIGHBOUR\_RSRP\_E** Neighbor cell RSRP.  
**QAPI\_DEVICE\_INFO\_NEIGHBOUR\_RSRQ\_E** Neighbor cell RSRQ.  
**QAPI\_DEVICE\_INFO\_NEIGHBOUR\_PCI\_E** Neighbor cell PCI.  
**QAPI\_DEVICE\_INFO\_NEIGHBOUR\_EARFCN\_E** Neighbor cell EARFCN.  
**QAPI\_DEVICE\_INFO\_TADV\_E** Device timing advance.  
**QAPI\_DEVICE\_INFO\_NETWORK\_IND\_E** Network indication.

### 20.1.3.2 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.



## 20.1.4 Function Documentation

### 20.1.4.1 `qapi_Status_t qapi_Device_Info_Init ( void )`

Initializes the device information context.

This function must be called before invoking other `qapi_Device_Info` APIs.

#### Returns

QAPI\_OK on success, QAPI\_ERROR on failure.

### 20.1.4.2 `qapi_Status_t qapi_Device_Info_Get ( qapi_Device_Info_ID_t id, qapi_Device_Info_t * info )`

Gets the device information for specified ID.

#### Parameters

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\(\)](#) must have been called.

### 20.1.4.3 `qapi_Status_t qapi_Device_Info_Set_Callback ( qapi_Device_Info_ID_t id, qapi_Device_Info_Callback_t callback )`

Sets a device information callback.

#### Parameters

in	<i>id</i>	Information ID.
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\(\)](#) must have been called.

#### 20.1.4.4 **qapi\_Status\_t qapi\_Device\_Info\_Release ( void )**

Releases the device information context.

##### **Returns**

QAPI\_OK on success, QAPI\_ERROR on failure.

##### **Dependencies**

Before calling this API, [qapi\\_Device\\_Info\\_Init\(\)](#) must have been called.

#### 20.1.4.5 **qapi\_Status\_t qapi\_Device\_Info\_Reset ( void )**

Resets the device.

##### **Returns**

QAPI\_OK on success, QAPI\_ERROR on failure.

## 20.2 Qapi\_net\_datatypes

The DNS (Domain Name System) 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, needed for the purpose of initiating communications with a remote server or service. The DNS service can be either manually configured, or automatically configured when the DHCP client is enabled.

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

# 21 Use Cases

---

This chapter provides a recommended API use case for the DSS APIs and an example of a socket API use case.

## 21.1 DSS API Use Case

This section provides a recommended DSS API procedure for successful data call establishment and tear down.

1. Ensure that the DSS library is initialized before using any DSS APIs. `dss_init()` must be invoked.
2. Always pass a callback function in `dss_get_data_srvc_hdl()` so that the call connection status is passed to the caller appropriately.
3. Copy the event in the callback function and switch the context for handling so that other clients will not be blocked.
4. Set all necessary parameters using `dss_set_data_call_param()`. A profile ID is recommended so that the modem does not pick up the default profile. Note that the DSS library does not do a profile look-up automatically.
5. Start the data call using the `dss_start_data_call()` API and expect either a `DSS_EVT_NET_IS_CONN` or `DSS_EVT_NET_NO_NET` event, where the former means success and the latter indicates a failure.
6. If the call is successful, fetch the IP address as follows:
  - (a) Fetch the number of addresses by calling `dss_get_ip_addr_count()`
  - (b) Call `dss_get_ip_addr()`
  - (c) Be sure to allocate the memory of the first parameter in `dss_get_ip_addr()` accordingly

## 21.2 Socket API Use Cases

This section provides a use case example for the Socket API that uses the loopback IP address 127.0.0.1 and port 5000 for the connection. The client and the server are expected to be running in two separate threads.

### 21.2.1 Server Socket

Initial socket settings:

```
int sock_fd = 0, errno = 0, new_sock_fd = 0;
unsigned short port = 5000;
char buf[1024] = "server_Hello";
```

```

    struct sockaddr_in server_addr, client_addr;

sock_fd = qapi_socket (AF_INET, SOCK_STREAM, 0);

```

Fetch a socket and bind it to the server IP address.

```

memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(port);
server_addr.sin_addr.s_addr = inet_addr("`127.0.0.1'");

if(qapi_bind(sock_fd, (struct sockaddr*)&server_addr, sizeof(struct
    sockaddr_in)) == -1)
{
    Printf("Address_binding_error\n");
}

qapi_setsockopt(sock_fd , SOL_SOCKET, SO_NBIO, NULL, 0);

if(qapi_listen(sock_ds, 5) == -1)

```

The above API marks the socket as a passive socket, which will be used to accept incoming connections. The second parameter indicates the queue length.

Let the server accept incoming connection requests.

```

new_sock_fd = qapi_accept(sock_fd, (struct sockaddr *)&client_addr, &len);

while(1)
{
    if( qapi_recv(new_sock_fd, buf, 128, 0) == -1)
        Printf("`qapi_server_recv failed \n'");
}

```

Upon accepting a connection, the server waits to receive data over the new socket file descriptor.

```

qapi_socketclose(new_sock_fd);
qapi_socketclose(sock_fd);

```

Once the data transfer is complete, terminate the connection by closing the file descriptors.

## 21.2.2 Client Socket

Initial socket settings:

```

int sock_ds= -1;
char buf[128];
struct sockaddr_in client_addr;
    unsigned short port = 5000;

sock_fd = qapi_socket(AF_INET, SOCK_STREAM, 0);
client_addr.sin_family = AF_INET;
client_addr.sin_port = htons(port);
client_addr.sin_addr.s_addr = inet_addr("`127.0.0.1'");
qapi_setsockopt(sock_ds , SOL_SOCKET, SO_NBIO, NULL, 0);

if(qapi_connect(sock_ds, (struct sockaddr*)&client_addr, sizeof(client_addr))
    == -1)
    Printf("Connect_failure(%s)\n", strerror(errno));

```

Connect to the server and send data.

```
if( qapi_send(sock_ds, buf, 128, 0) == -1)
    Printf("send_failed\n");
```

Once done, close the socket.

```
qapi_socketclose(sock_ds);
```

QTI recommends that the socket data transmissions occur every 10 msec or more. Also, the priority of threads for client and server should be more than 150.

This networking stack has a smaller reserved memory, so be sure to handle an ENOBUFS socket error that indicates an out of memory condition.

## 21.3 TLS/DTLS API Use Cases

This section provides use case examples for the SSL API that use the loopback IP address 127.0.0.1 and port 5000 for the connection. The client is expected to be running in two separate threads (one for sending data and one for receiving data).

```
/* TLS/DTLS Instance structure */
typedef struct ssl_inst
{
    qapi_Net_SSL_Obj_Hdl_t sslCtx;
    qapi_Net_SSL_Con_Hdl_t sslCon;
    qapi_Net_SSL_Config_t config;
    qapi_Net_SSL_Role_t role;
} SSL_INST;
```

### 21.3.1 TLS/DTLS Context Object Creation

```
SSL_INST ssl;

/* TLS Client object creation. */
qapi_Net_SSL_Role_t role = QAPI_NET_SSL_CLIENT_E;

memset(&ssl, 0, sizeof(SSL_INST));
ssl.role = role;
ssl.sslCtx = qapi_Net_SSL_Obj_New(role);

if (ssl.sslCtx == QAPI_NET_SSL_INVALID_HANDLE)
{
    printf("ERROR:_Unable_to_create_SSL_context");
    return QCLI_STATUS_ERROR_E;
}
```

## 21.3.2 TLS/DTLS Certificate or CA List, or PSK Table Store and Load to SSL Context

```

char *cert_data_buf;
int cert_data_buf_len;

/* allocate memory and read the certificate from certificate server or EFS.
Once
cert_data_buf filled with valid SSL certificate, Call QAPI to Store and
Load */

/* Store and Loading Certificate */
char * name = "Sample_cert.bin";
result = qapi_Net_SSL_Cert_Store(name, QAPI_NET_SSL_CERTIFICATE_E,
cert_data_buf, cert_data_buf_len);
if (result == QAPI_OK)
{
    if (qapi_Net_SSL_Cert_Load(ssl.sslCtx, QAPI_NET_SSL_CERTIFICATE_E, name) <
        0)
    {
        printf("ERROR:_Unable_to_load_%s_from_FLASH\r\n" , name);
        return QCLI_STATUS_ERROR_E;
    }
}

/* Store and CA List(Root Certificates) */
char * name = "Sample_cert.bin";
result = qapi_Net_SSL_Cert_Store(name, QAPI_NET_SSL_CA_LIST_E, cert_data_buf,
cert_data_buf_len);
if (result == QAPI_OK)
{
    if (qapi_Net_SSL_Cert_Load(ssl.sslCtx, QAPI_NET_SSL_CA_LIST_E, name) < 0)
    {
        printf("ERROR:_Unable_to_load_%s_from_FLASH\r\n" , name);
        return QCLI_STATUS_ERROR_E;
    }
}

```

## 21.3.3 TLS/DTLS Connection Object Creation

```

printf("Create_new_TLS_Connection");
ssl.sslCon = qapi_Net_SSL_Con_New(ssl.sslCtx, QAPI_NET_SSL_TLS_E);
if (ssl.sslCon == QAPI_NET_SSL_INVALID_HANDLE)
{
    printf("ERROR:_Unable_to_create_SSL_context");
    return QCLI_STATUS_ERROR_E;
}

```

## 21.3.4 TLS/DTLS Configuration of a Connection Object

```

ssl.config.cipher[0] = QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA;
ssl.config.max_Frag_Len = 4096;
ssl.config.max_Frag_Len_Neg_Disable = 0;
ssl.config.protocol = TLS1.2
ssl.config.verify.domain = 0;

```

```

ssl.config.verify.match_Name[0] = '\\0';
ssl.config.verify.send_Alert = 0;
ssl.config.verify.time_Validity = 1;

result = qapi_Net_SSL_Configure(ssl.sslCon, &ssl.onfig);
if (result < QAPI_OK)
{
    printf("ERROR:_SSL_configure_failed_(%d)", result);
    return QCLI_STATUS_ERROR_E;
}

```

## 21.3.5 Secure Socket Data Transfer over a TLS/DTLS Connection

### Initial socket setting

```

int sock_ds= -1;
char buf[128];
char recvbuf[128];

struct sockaddr_in client_addr;
unsigned short port = 5000;

sock_fd = qapi_socket(AF_INET, SOCK_STREAM, 0);

client_addr.sin_family = AF_INET;
client_addr.sin_port = htons(port);
client_addr.sin_addr.s_addr = inet_addr('\\127.0.0.1');

qapi_setsockopt(sock_ds, SOL_SOCKET, SO_NBIO, NULL, 0);

if(qapi_connect(sock_ds, (struct sockaddr*)&client_addr, sizeof(client_addr))
    == -1)
    Printf("Connect_failure(%s)", strerror(errno));

printf("SSL_Connecting");

```

### Attach a socket handle with a TLS/DTLS connection

```

printf("Add_socket_handle_to_SSL_connection");
result = qapi_Net_SSL_Fd_Set(ssl.sslCon, sock_ds);
if (result < 0)
{
    printf("ERROR:_Unable_to_add_socket_handle_to_SSL_(%d)", result);
    goto ERROR;
}

```

### Initiate the TLS/DTLS handshake

```

printf("Start_TLS/DTLS_handshake_with_server");
result = qapi_Net_SSL_Connect(ssl.sslCon);

app_msec_delay(10);

if (result < 0)
{
    if (result == QAPI_SSL_OK_HS)

```



```

{
    /** The peer's SSL certificate is trusted, CN matches the host name,
        time is valid */
    printf("The_certificate_is_trusted");
}
else if (result == QAPI_ERR_SSL_CERT_CN)
{
    /** The peer's SSL certificate is trusted, CN matches the host name,
        time is expired */
    printf("ERROR:_The_certificate_is_expired");
    goto ERROR;
}
else if (result == QAPI_ERR_SSL_CERT_TIME)
{
    /** The peer's SSL certificate is trusted, CN does NOT match the host
        name, time is valid */
    printf(qcli_net_handle, "ERROR:_The_certificate_is_trusted,_but_the_
        host_name_is_not_valid");
    goto ERROR;
}
else if (result == QAPI_ERR_SSL_CERT_NONE)
{
    /** The peer's SSL certificate is trusted, CN does NOT match host name
        , time is expired */
    printf("ERROR:_The_certificate_is_expired_and_the_host_name_is_not_
        valid");
    goto ERROR;
}
else
{
    printf("ERROR:_SSL_connect_failed_(%d)", result);
    goto ERROR;
}
}

```

### Send/receive secure data over a TLS/DTLS connection

```
qapi_Net_SSL_Write(ssl.sslCon, buf, 128,);
```

To receive data on the same SSL Session, user need to create a recv thread and use the same SSL connection Descriptor.

```
qapi_Net_SSL_Read(ssl.sslCon, recvbuf, 128);
```

### 21.3.6 Close an SSL Connection TLS/DTLS Connection and Socket

```

ERROR:
if (role == QAPI_NET_SSL_CLIENT_E &&  ssl.sslCon !=
    QAPI_NET_SSL_INVALID_HANDLE)
{
    qapi_Net_SSL_Shutdown(ssl.sslCon);
    ssl.sslCon = QAPI_NET_SSL_INVALID_HANDLE;
}

qapi_socketclose(sock_ds);

```

### 21.3.7 TLS/DTLS Close Context Object

```
if (ssl.sslCtx)
{
    qapi_Net_SSL_Obj_Free(ssl.sslCtx);
    ssl, sslCtx = QAPI_NET_SSL_INVALID_HANDLE;
}
```

QUALCOMM®  
2017-05-07 17:57:37 PDT  
hyman.ding@qtecel.com

# A References

---

## A.1 Related Documents

Title	Number
<b>Qualcomm Technologies</b>	
<i>MDM9206 Data Features Overview</i>	80-P8101-7

## A.2 Acronyms and Terms

Acronym or term	Definition
APN	Access point name
BSD	Berkeley Software Distribution
CA	Certificate authority
CE	Call end
DHCP	Dynamic Host Configuration Protocol
DNS	Domain name or system
DSS	Data services sockets
MTU	Maximum transmission unit
netctrl	Net control
PDP	Packet Data Protocol
PSK	Preshared key
QAPI	Qualcomm API
QMI	Qualcomm messaging interface
SPI	Serial peripheral interface