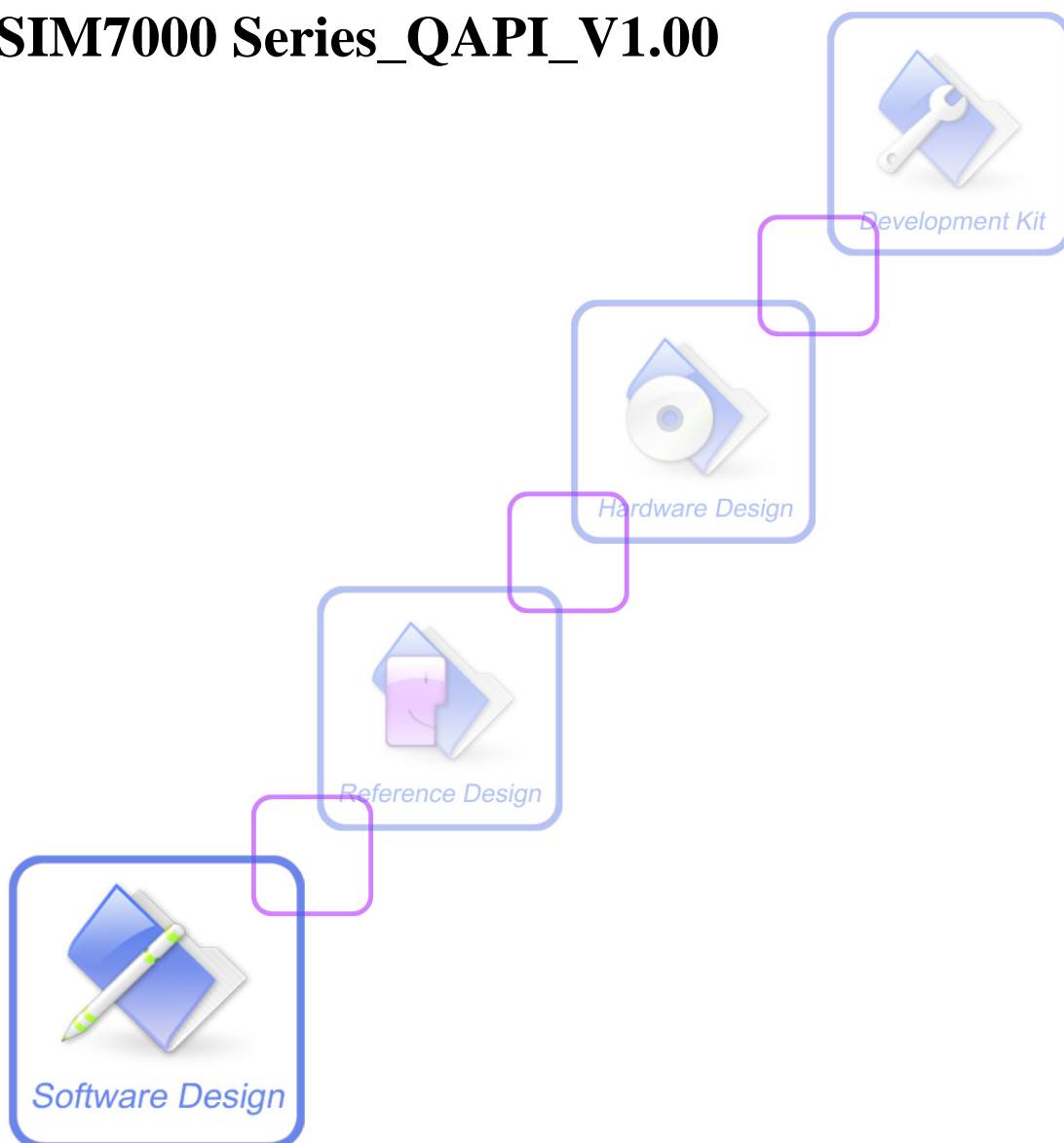




SIM7000 Series_QAPI_V1.00



Document Title	SIM7000 Series QAPI
Version	1.00
Date	2017-11-27
Status	Release
Document Control ID	SIM7000 Series_QAPI_V1.00

General Notes

SIMCom offers this information as a service to its customers, to support application and engineering efforts that use the products designed by SIMCom. The information provided is based upon requirements specifically provided to SIMCom by the customers. SIMCom has not undertaken any independent search for additional relevant information, including any information that may be in the customer's possession. Furthermore, system validation of this product designed by SIMCom within a larger electronic system remains the responsibility of the customer or the customer's system integrator. All specifications supplied herein are subject to change.

Copyright

This document contains proprietary technical information which is the property of Shanghai SIMCom Wireless Solutions Ltd, copying of this document and giving it to others and the using or communication of the contents thereof, are forbidden without express authority. Offenders are liable to the payment of damages. All rights reserved in the event of grant of a patent or the registration of a utility model or design. All specification supplied herein are subject to change without notice at any time.

Copyright © Shanghai SIMCom Wireless Solutions Ltd. 2017

Contents

1	INTRODUCTION.....	19
1.1	PURPOSE.....	19
1.2	CONVENTIONS.....	19
2	DATA CALL FUNCTIONAL OVERVIEW	20
2.1	DATA CALL ARCHITECTURE IN THE THREADX OS.....	20
3	DSS NET CONTROL APIS	21
3.1	DSS NETCTRL MACROS, DATA STRUCTURES, AND ENUMERATIONS	22
3.1.1	<i>Define Documentation</i>	<i>23</i>
3.1.1.1	#define QAPI_DSS_RADIO_TECH_UNKNOWN 0x00000000	23
3.1.1.2	#define QAPI_DSS_RADIO_TECH_MIN 0x00000001.....	23
3.1.1.3	#define QAPI_DSS_RADIO_TECH_UMTS QAPI_DSS_RADIO_TECH_MIN.....	23
3.1.1.4	#define QAPI_DSS_RADIO_TECH_CDMA 0x00000002	23
3.1.1.5	#define QAPI_DSS_RADIO_TECH_1X 0x00000004.....	23
3.1.1.6	#define QAPI_DSS_RADIO_TECH_DO 0x00000008.....	23
3.1.1.7	#define QAPI_DSS_RADIO_TECH_LTE 0x00000010	23
3.1.1.8	#define QAPI_DSS_RADIO_TECH_TDSCDMA 0x00000020	23
3.1.1.9	#define QAPI_DSS_CALL_INFO_USERNAME_MAX_LEN 127	23
3.1.1.10	#define QAPI_DSS_CALL_INFO_PASSWORD_MAX_LEN 127	23
3.1.1.11	#define QAPI_DSS_CALL_INFO_APN_MAX_LEN 150.....	23
3.1.1.12	#define QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN 12	24
3.1.1.13	#define QAPI_DSS_SUCCESS 0.....	24
3.1.1.14	#define QAPI_DSS_ERROR1	24
3.1.1.15	#define QAPI_DSS_IP_VERSION_44	24
3.1.1.16	#define QAPI_DSS_IP_VERSION_66	24
3.1.1.17	#define QAPI_DSS_IP_VERSION_4_610	24
3.1.1.18	#define QAPI_DSS_MODE_GENERAL0	24
3.1.2	<i>Data Structure Documentation</i>	<i>24</i>
3.1.2.1	struct qapi_DSS_CE_Reason_s	24
3.1.2.2	struct qapi_DSS_Call_Param_Value_s	24
3.1.2.3	struct qapi_DSS_Addr_s.....	25
3.1.2.4	union qapi_DSS_Addr_s::qapi_dss_ip_address_u.....	25
3.1.2.5	struct qapi_DSS_Addr_Info_s	25
3.1.2.6	struct qapi_DSS_Data_Pkt_Stats_s.....	25
3.1.3	<i>Typedef Documentation</i>	<i>26</i>
3.1.3.1	qapi_DSS_Net_Ev_CB_t	26
3.1.4	<i>Enumeration Type Documentation.....</i>	<i>26</i>
3.1.4.1	enum qapi_DSS_Auth_Pref_e.....	26
3.1.4.2	enum qapi_DSS_CE_Reason_Type_e.....	26
3.1.4.3	enum qapi_DSS_Call_Info_Enum_e.....	27
3.1.4.4	enum qapi_DSS_Net_Evt_e	27
3.1.4.5	enum qapi_DSS_IP_Family_e	27
3.1.4.6	enum qapi_DSS_Data_Bearer_Tech_e	27
3.1.4.7	enum qapi_DSS_Call_Tech_Type_e.....	28
3.2	INITIALIZE THE DSS NETCTRL LIBRARY	28
3.2.1	<i>Function Documentation.....</i>	<i>28</i>
3.2.1.1	qapi_Status_t qapi_DSS_Init (int mode)	28
3.3	RELEASE THE DSS NETCTRL LIBRARY.....	29
3.3.1	<i>Function Documentation.....</i>	<i>29</i>
3.3.1.1	qapi_Status_t qapi_DSS_Release (int mode)	29
3.4	GET THE DATA SERVICE HANDLE	29
3.4.1	<i>Function Documentation.....</i>	<i>29</i>
3.4.1.1	qapi_Status_t qapi_DSS_Get_Data_Srvc_Hndl.....	29
3.5	RELEASE THE DATA SERVICE HANDLE.....	30
3.5.1	<i>Function Documentation.....</i>	<i>30</i>
3.5.1.1	qapi_Status_t qapi_DSS_Rel_Data_Srvc_Hndl (qapi_DSS_Hndl_t hndl).....	30
3.6	SET THE DATA CALL PARAMETER.....	30

3.6.1	Function Documentation.....	30
3.6.1.1	qapi_Status_t qapi_DSS_Set_Data_Call_Param	30
3.7	START A DATA CALL	31
3.7.1	Function Documentation.....	31
3.7.1.1	qapi_Status_t qapi_DSS_Start_Data_Call(qapi_DSS_Hndl_t hndl)	31
3.8	STOP A DATA CALL	31
3.8.1	Function Documentation.....	31
3.8.1.1	qapi_Status_t qapi_DSS_Stop_Data_Call(qapi_DSS_Hndl_t hndl)	31
3.9	GET PACKET DATA TRANSFER STATISTICS.....	32
3.9.1	Function Documentation.....	32
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)	32
3.10	RESET PACKET DATA TRANSFER STATISTICS	33
3.10.1	Function Documentation.....	33
3.10.1.1	qapi_Status_t qapi_DSS_Reset_Pkt_Stats(qapi_DSS_Hndl_t hndl)	33
3.11	GET THE DATA CALL END REASON	33
3.11.1	Function Documentation.....	33
3.11.1.1	qapi_Status_t qapi_DSS_Get_Call_End_Reason.....	33
3.12	GET THE DATA CALL TECHNOLOGY	34
3.12.1	Function Documentation.....	34
3.12.1.1	qapi_Status_t qapi_DSS_Get_Call_Tech.....	34
3.13	GET THE DATA BEARER TECHNOLOGY	34
3.13.1	Function Documentation.....	34
3.13.1.1	qapi_Status_t qapi_DSS_Get_Current_Data_Bearer_Tech.....	34
3.14	GET THE DEVICE NAME.....	35
3.14.1	Function Documentation.....	35
3.14.1.1	qapi_Status_t qapi_DSS_Get_Device_Name(qapi_DSS_Hndl_t hndl, char *buf, int len)	35
3.15	GET THE QMI PORT NAME.....	35
3.15.1	Function Documentation.....	35
3.15.1.1	qapi_Status_t qapi_DSS_Get_Qmi_Port_Name(qapi_DSS_Hndl_t hndl, charbuf, int len)	35
3.16	GET THE IP ADDRESS COUNT	36
3.16.1	Function Documentation.....	36
3.16.1.1	qapi_Status_t qapi_DSS_Get_IP_Addr_Count(qapi_DSS_Hndl_t hndl, unsigned int * ip_addr_cnt).....	36
3.17	GET THE IP ADDRESS INFORMATION	36
3.17.1	Function Documentation.....	36
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).....	36
3.18	GET THE IP ADDRESS INFORMATION STRUCTURE.....	37
3.18.1	Function Documentation.....	37
3.18.1.1	qapi_Status_t qapi_DSS_Get_IP_Addr_Per_Family	37
3.19	GET THE LINK MTU INFORMATION.....	38
3.19.1	Function Documentation.....	38
3.19.1.1	qapi_Status_t qapi_DSS_Get_Link_Mtu(qapi_DSS_Hndl_t hndl, unsigned int *mtu)....	38
4	QAPI NETWORKING SOCKET.....	39
4.1	QAPI SOCKET MACROS AND DATA STRUCTURES	40
4.1.1	Define Documentation	43
4.1.2	Data Structure Documentation	51
4.1.2.1	struct in_addr	51
4.1.2.2	struct sockaddr_in	51
4.1.2.3	struct in6_addr	51
4.1.2.4	struct ip46addr_n	52
4.1.2.5	union ip46addr_n.a	52
4.1.2.6	union ip46addr_n.g	52
4.1.2.7	struct sockaddr_in6	52
4.1.2.8	struct ip46addr	52

4.1.2.9	union ip46addr	53
4.1.2.10	struct sockaddr	53
4.1.2.11	struct fd_set	53
4.2	CREATE A SOCKET	53
4.2.1	Function Documentation	53
4.2.1.1	int qapi_socket(int32_t family, int32_t type, int32_t protocol)	53
4.3	BIND A SOCKET	54
4.3.1	Function Documentation	54
4.3.1.1	qapi_Status_t qapi_bind(int32_t handle, struct sockaddr *addr, int32_t addrlen)	54
4.4	MAKE A SOCKET PASSIVE	54
4.4.1	Function Documentation	54
4.4.1.1	qapi_Status_t qapi_listen(int32_t handle, int32_t backlog)	54
4.5	ACCEPT A SOCKET CONNECTION REQUEST	55
4.5.1	Function Documentation	55
4.5.1.1	int qapi_accept(int32_t handle, struct sockaddr *cliaddr, int32_t *addrlen)	55
4.6	CONNECT TO A SOCKET	55
4.6.1	Function Documentation	55
4.6.1.1	qapi_Status_t qapi_connect(int32_t handle, struct sockaddr *srvaddr,	55
4.7	SET SOCKET OPTIONS	56
4.7.1	Function Documentation	56
4.7.1.1	qapi_Status_t qapi_setsockopt(int32_t handle, int32_t level, int32_t	56
4.8	GET SOCKET OPTIONS	56
4.8.1	Function Documentation	56
4.8.1.1	qapi_Status_t qapi_getsockopt(int32_t handle, int32_t level, int32_t optname, void *optval, int32_t optlen)	56
4.9	CLOSE A SOCKET	57
4.9.1	Function Documentation	57
4.9.1.1	qapi_Status_t qapi_socketclose(int32_t handle)	57
4.10	GET A SOCKET ERROR CODE	57
4.10.1	Function Documentation	57
4.10.1.1	int qapi_errno(int32_t handle)	57
4.11	RECEIVE A MESSAGE FROM A SOCKET	57
4.11.1	Function Documentation	57
4.11.1.1	int qapi_recvfrom	57
4.12	RECEIVE A MESSAGE FROM A CONNECTED SOCKET	58
4.12.1	Function Documentation	58
4.12.1.1	int qapi_recv(int32_t handle, char *buf, int32_t len, int32_t flags)	58
4.13	SEND A MESSAGE ON A SOCKET	59
4.13.1	Function Documentation	59
4.13.1.1	int qapi_sendto(int32_t handle, char *buf, int32_t len, int32_t flags, struct sockaddr *to, int32_t tolen)	59
4.14	SEND A MESSAGE ON A CONNECTED SOCKET	60
4.14.1	Function Documentation	60
4.14.1.1	int qapi_send(int32_t handle, char *buf, int32_t len, int32_t flags)	60
4.15	SELECT A SOCKET	60
4.15.1	Function Documentation	60
4.15.1.1	int qapi_select(fd_set *rd, fd_set *wr, fd_set *ex, int32_t timeout_ms)	60
4.16	INITIALIZE A SOCKET	61
4.16.1	Function Documentation	61
4.16.1.1	qapi_Status_t qapi_fd_zero(fd_set *set)	61
4.17	CLEAR A SOCKET FROM A SOCKET SET	61
4.17.1	Function Documentation	61
4.17.1.1	qapi_Status_t qapi_fd_clr(int32_t handle, fd_set *set)	61
4.18	ADD A SOCKET TO A SOCKET SET	61
4.18.1	Function Documentation	61
4.18.1.1	qapi_Status_t qapi_fd_set(int32_t handle, fd_set *set)	61
4.19	CHECK WHETHER A SOCKET IS IN A SOCKET SET	62
4.19.1	Function Documentation	62
4.19.1.1	qapi_Status_t qapi_fd_isset(int32_t handle, fd_set *set)	62

4.20	GET THE ADDRESS OF A CONNECTED PEER	62
4.20.1	<i>Function Documentation</i>	62
	qapi_Status_t qapi_getpeername(int32_t handle, struct sockaddr *addr, int *addrlen).....	62
4.21	GET THE ADDRESS TO WHICH THE SOCKET IS BOUND	63
4.21.1	<i>Function Documentation</i>	63
4.21.1.1	qapi_Status_t qapi_getsockname(int32_t handle, struct sockaddr *addr, int *addrlen).....	63
5	QAPI NETWORK SECURITY APIS	64
5.1	QAPI SSL DATA TYPES.....	64
5.1.1	<i>Define Documentation</i>	65
5.1.2	<i>Data Structure Documentation</i>	71
5.1.2.1	struct qapi_Net_SSL_Verify_Policy_s	71
5.1.2.2	struct qapi_Net_SSL_Config_s.....	71
5.1.2.3	struct qapi_Net_SSL_Cert_List_s.....	71
5.1.2.4	struct qapi_Net_SSL_CERT_s	72
5.1.2.5	struct qapi_Net_SSL_CA_Info_s.....	72
5.1.2.6	struct qapi_Net_SSL_CA_List_s	72
5.1.2.7	struct qapi_Net_SSL_PSK_Table_s	72
5.1.2.8	struct qapi_Net_SSL_Cert_Info_s.....	73
5.1.2.9	union qapi_Net_SSL_Cert_Info_s.info	73
5.1.3	<i>Enumeration Type Documentation</i>	73
5.1.3.1	enum qapi_Net_SSL_Role_t	73
5.1.3.2	enum qapi_Net_SSL_Protocol_t	73
5.1.3.3	enum qapi_Net_SSL_Cert_Type_t	74
5.2	QAPI SSL TYPEDEFS.....	74
5.2.1	<i>Typedef Documentation</i>	74
5.2.1.1	typedef uint32_t qapi_Net_SSL_Obj_Hdl_t	74
5.2.1.2	typedef uint32_t qapi_Net_SSL_Con_Hdl_t.....	74
5.2.1.3	typedef const void* qapi_Net_SSL_Cert_t.....	74
5.2.1.4	typedef const void* qapi_Net_SSL_CAList_t.....	74
5.2.1.5	typedef const void* qapi_Net_SSL_PSKTable_t.....	75
5.3	CREATE AN SSL OBJECT.....	75
5.3.1	<i>Function Documentation</i>	75
5.3.1.1	qapi_Net_SSL_Obj_Hdl_t qapi_Net_SSL_Obj_New(qapi_Net_SSL_Role_t role).....	75
5.4	CREATE AN SSL CONNECTION HANDLE	75
5.4.1	<i>Function Documentation</i>	75
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)	75
5.5	CONFIGURE AN SSL CONNECTION	76
5.5.1	<i>Function Documentation</i>	76
5.5.1.1	qapi_Status_t qapi_Net_SSL_Configure(qapi_Net_SSL_Con_Hdl_t ssl, qapi_Net_SSL_Config_t *cfg).....	76
5.6	DELETE AN SSL CERTIFICATE	77
5.6.1	<i>Function Documentation</i>	77
5.6.1.1	qapi_Status_t qapi_Net_SSL_Cert_delete(char *name, qapi_Net_SSL_Cert_Type_t type) 77	
5.7	STORE AN SSL CERTIFICATE.....	77
5.7.1	<i>Function Documentation</i>	77
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)	77
5.8	CONVERT AND STORE AN SSL CERTIFICATE.....	78
5.8.1	<i>Function Documentation</i>	78
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)	78
5.9	LOAD AN SSL CERTIFICATE.....	79
5.9.1	<i>Function Documentation</i>	79
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).....	79
5.10	GET A LIST OF SSL CERTIFICATES.....	79
5.10.1	<i>Function Documentation</i>	79

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)	79
5.11	ATTACH A SOCKET DESCRIPTOR TO THE SSL CONNECTION	80
5.11.1	Function Documentation.....	80
5.11.1.1	qapi_Status_t qapi_Net_SSL_Fd_Set(qapi_Net_SSL_Con_Hdl_t ssl, uint32_t fd)	80
5.12	ACCEPT AN SSL CONNECTION FROM THE CLIENT.....	80
5.12.1	Function Documentation.....	80
5.12.1.1	qapi_Status_t qapi_Net_SSL_Accept(qapi_Net_SSL_Con_Hdl_t ssl)	80
5.13	INITIATE AN SSL HANDSHAKE	81
5.13.1	Function Documentation.....	81
5.13.1.1	qapi_Status_t qapi_Net_SSL_Connect(qapi_Net_SSL_Con_Hdl_t ssl)	81
5.14	CLOSE AN SSL CONNECTION	81
5.14.1	Function Documentation.....	81
5.14.1.1	qapi_Status_t qapi_Net_SSL_Shutdown(qapi_Net_SSL_Con_Hdl_t ssl)	81
5.15	FREE AN SSL OBJECT HANDLE	81
5.15.1	Function Documentation.....	81
5.15.1.1	qapi_Status_t qapi_Net_SSL_Obj_Free(qapi_Net_SSL_Obj_Hdl_t hdl)	81
5.16	READ SSL DATA.....	82
5.16.1	Function Documentation.....	82
5.16.1.1	qapi_Status_t qapi_Net_SSL_Read(qapi_Net_SSL_Con_Hdl_t hdl, void *buf, uint32_t size)	82
5.17	WRITE SSL DATA	82
5.17.1	Function Documentation.....	82
5.17.1.1	qapi_Status_t qapi_Net_SSL_Write(qapi_Net_SSL_Con_Hdl_t hdl, void *buf, uint32_t size)	82
6	QAPI NETWORKING SERVICES.....	84
6.1	NETWORKING SERVICES MACROS, DATA TYPES, AND ENUMERATIONS	84
6.1.1	Define Documentation	84
6.1.1.1	#define QAPI_IPV4_IS_MULTICAST(ipv4_Address) (((long)(ipv4_Address) & 0xf0000000) == 0xe0000000)	84
6.1.1.2	#define IF_NAMELEN 20	85
6.1.1.3	#define QAPI_NET_IPV4_MAX_ROUTES (3).....	85
6.1.1.4	#define QAPI_IS_IPV6_LINK_LOCAL(ipv6_Address)	85
6.1.1.5	#define QAPI_IS_IPV6_MULTICAST(ipv6_Address) (ipv6_Address[0] == 0xff).....	85
6.1.1.6	#define QAPI_NET_IPV6_MAX_ROUTES (3).....	85
6.1.1.7	#define QAPI_NET_IFNAME_LEN 12	86
6.1.2	Data Structure Documentation	86
6.1.2.1	struct qapi_Net_Ping_V4_s	86
6.1.2.2	struct qapi_Net_IPv4_Route_t	86
6.1.2.3	struct qapi_Net_IPv4_Route_List_t	86
6.1.2.4	struct qapi_Net_Ping_V6_s	87
6.1.2.5	struct qapi_Net_IPv6_Route_t	87
6.1.2.6	struct qapi_Net_IPv6_Route_List_t	87
6.1.2.7	struct qapi_Net_Ifnameindex_t	87
6.1.2.8	struct qapi_Ping_Info_Resp_s	88
6.1.3	Enumeration Type Documentation.....	88
6.1.3.1	enum qapi_Net_Route_Command_t.....	88
6.2	GET THE NAMES OF ALL NETWORK INTERFACES	88
6.2.1	Function Documentation.....	88
6.2.1.1	int32_t qapi_Net_Get_All_Ifnames(qapi_Net_Ifnameindex_t *ifname_index)	88
6.3	PARSE AN ADDRESS STRING INTO AN IPV4/IPV6 ADDRESS.....	89
6.3.1	Function Documentation.....	89
6.3.1.1	int32_t inet_pton(int32_t af, const char *src, void *dst)	89
6.4	FORMAT AN IPV4/IPV6 ADDRESS INTO A NULL-TERMINATED STRING	89
6.4.1	Function Documentation.....	89
6.4.1.1	const char *inet_ntop(int32_t af, const void *src, char *dst, size_t size)	89
6.5	GET THE PHYSICAL ADDRESS AND LENGTH OF AN INTERFACE.....	89
6.5.1	Function Documentation.....	89
6.5.1.1	int32_t qapi_Net_Interface_Get_Physical_Address(const char * interface_Name, const	

uint8_t ** address, uint32_t * address_Len).....	89
6.6 CHECK WHETHER AN INTERFACE EXISTS.....	90
6.6.1 Function Documentation.....	90
6.6.1.1 qbool_t qapi_Net_Interface_Exist(const char * interface_Name).....	90
6.7 IPV4 NETWORK CONFIGURATION.....	91
6.7.1 Function Documentation.....	91
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).....	91
6.8 SEND AN IPV4 PING.....	91
6.8.1 Function Documentation.....	91
6.8.1.1 qapi_Status_t qapi_Net_Ping(uint32_t ipv4_Addr, uint32_t size).....	91
6.9 SEND AN IPV4 PING WITH A RESPONSE.....	91
6.9.1 Function Documentation.....	91
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).....	91
6.10 IPV4 ROUTE COMMANDS.....	92
6.10.1 Function Documentation.....	92
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).....	92
6.11 SEND AN IPV6 PING.....	92
6.11.1 Function Documentation.....	92
6.11.1.1 qapi_Status_t qapi_Net_Ping6(uint8_t ipv6_Addr[16], uint32_t size, const char * interface_Name).....	92
6.12 SEND AN IPV6 PING WITH A RESONSE.....	93
6.12.1 Function Documentation.....	93
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).....	93
6.13 GET THE IPV6 ADDRESS OF AN INTERFACE.....	93
6.13.1 Function Documentation.....	93
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).....	93
6.14 IPV6 ROUTE COMMANDS.....	94
6.14.1 Function Documentation.....	94
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).....	94
6.15 GET THE INTERFACE SCOPE ID.....	95
6.15.1 Function Documentation.....	95
qapi_Status_t qapi_Net_IPv6_Get_Scope_ID(const char * interface_Name, int32_t * cope_ID).....	95
7 DOMAIN NAME SYSTEM CLIENT SERVICE APIS.....	96
7.1 DNS CLIENT SERVICE MACROS, DATA TYPES, AND ENUMERATIONS.....	96
7.1.1 Define Documentation.....	96
7.1.2 Data Structure Documentation.....	97
7.1.2.1 struct qapi_Net_DNS_Server_List_t.....	97
7.1.2.2 struct qapi_hostent_s.....	97
7.1.3 Enumeration Type Documentation.....	97
7.2 CHECK WHETHER THE DNS CLIENT HAS STARTED.....	98
7.2.1 Function Documentation.....	98
7.2.1.1 int32_t qapi_Net_DNSc_Is_Started(void).....	98
7.3 START, STOP, OR DISABLE THE DNS CLIENT.....	98
7.3.1 Function Documentation.....	98
7.3.1.1 int32_t qapi_Net_DNSc_Command(qapi_Net_DNS_Command_t cmd).....	98
7.4 CONVERT AN IP ADDRESS TEXT STRING INTO AN IP ADDRESS.....	98
7.4.1 Function Documentation.....	98

7.4.1.1	int32_t qapi_Net_DNSc_Reshost (char *hostname, struct ip46addr *ipaddr).....	98
7.5	CONVERT AN IP ADDRESS TEXT STRING FOR AN INTERFACE	99
7.5.1	Function Documentation.....	99
7.6	GET A LIST OF DNS SERVERS.....	99
7.6.1	Function Documentation.....	99
7.6.1.1	int32_t qapi_Net_DNSc_Get_Server_List (qapi_Net_DNS_Server_List_t *svr_list, uint8_t iface_index).....	99
7.7	GET INDEX FOR ADDED DNS SERVER	99
7.7.1	Function Documentation.....	99
7.7.1.1	qapi_Status_t qapi_Net_DNSc_Get_Server_Index (char *svr_addr, uint32_t *id, char *iface) 100	
7.8	ADD A DNS SERVER	100
7.8.1	Function Documentation.....	100
7.8.1.1	int32_t qapi_Net_DNSc_Add_Server(char*svr_addr, uint32_t id)	100
7.9	ADD A DNS SERVER TO AN INTERFACE	100
7.9.1	Function Documentation.....	100
7.9.1.1	int32_t qapi_Net_DNSc_Add_Server_on_iface(char*svr_addr, uint32_t id,char *iface) 100	
7.10	REMOVE A DNS SERVER	101
7.10.1	Function Documentation.....	101
7.10.1.1	int32_t qapi_Net_DNSc_Del_Server(uint32_t id).....	101
7.11	REMOVES A DNS SERVER FROM AN INTERFACE	101
7.11.1	Function Documentation.....	101
7.11.1.1	int32_t qapi_Net_DNSc_Del_Server_on_iface(uint32_t id, char*iface_index).....	101
7.12	GET IPV4 HOST INFORMATION BY NAME	101
7.12.1	Function Documentation.....	101
7.12.1.1	struct qapi_hostent_s*qapi_Net_DNSc_Get_Host_By_Name(char*name)[read]	101
7.13	GET IPV4/IPV6 HOST INFORMATION BY NAME.....	102
7.13.1	Function Documentation.....	102
7.13.1.1	7.13.1.1 struct qapi_hostent_s* qapi_Net_DNSc_Get_Host_By_Name2 (char *name,uint32_t af) [read].....	102
8	HTTP(S) API	103
8.1	HTTP(S) API	103
8.1.1	Data Structure Documentation	103
8.1.1.1	struct qapi_Net_HTTPc_Response_t	103
8.1.2	Typedef Documentation	103
8.1.2.1	typedef void(* qapi_HTTPc_CB_t)(void *arg, int32_t state, void *value)	103
8.1.3	Enumeration Type Documentation.....	103
8.1.3.1	enum qapi_Net_HTTPc_Method_e	103
8.1.3.2	enum qapi_Net_HTTPc_CB_State_e.....	104
8.1.4	Function Documentation.....	104
8.1.4.1	qapi_Status_t qapi_Net_HTTPc_Start(void).....	104
8.1.4.2	qapi_Status_t qapi_Net_HTTPc_Stop(void)	104
8.1.4.3	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)	105
8.1.4.4	qapi_Status_t qapi_Net_HTTPc_Free_sess(qapi_Net_HTTPc_handle_t handle)	105
8.1.4.5	qapi_Status_t qapi_Net_HTTPc_Connect(qapi_Net_HTTPc_handle_t handle,const char *URL, uint16_t port).....	105
8.1.4.6	qapi_Status_t qapi_Net_HTTPc_Disconnect(qapi_Net_HTTPc_handle_t handle)	106
8.1.4.7	qapi_Status_t qapi_Net_HTTPc_Request(qapi_Net_HTTPc_handle_t handle,qapi_Net_HTTPc_Method_e cmd, const char *URL)	106
8.1.4.8	qapi_Status_t qapi_Net_HTTPc_Set_Body(qapi_Net_HTTPc_handle_t handle, const char *body, uint32_t body_Length)	106
8.1.4.9	qapi_Status_t qapi_Net_HTTPc_Set_Param (qapi_Net_HTTPc_handle_t handle, const char *key, const char *value)	107
8.1.4.10	qapi_Status_t qapi_Net_HTTPc_Add_Header_Field(qapi_Net_HTTPc_handle_t handle, const char *type, const char *value)	107
8.1.4.11	qapi_Status_t qapi_Net_HTTPc_Clear_Header(qapi_Net_HTTPc_handle_t handle)	107

8.1.4.12	qapi_Status_tqapi_Net_HTTPC_Configure_SSL(qapi_Net_HTTPC_handle_t handle, qapi_Net_SSL_Config_t * ssl_Cfg)	108
9	QAPI STATUS AND ERROR CODES	109
9.1	QAPI STATUS CODES	109
9.1.1	SSL Module Error Codes	109
9.1.2	Generic Error Codes	109
9.1.3	MQTT Error Codes	109
9.1.4	QAPI Modules	111
9.1.5	Common QAPI Status Codes	112
9.2	DEFINE DOCUMENTATION	113
10	SYSTEM DRIVERS APIS	120
10.1	GPIO INTERRUPT CONTROLLER APIS	120
10.1.1	Typedef Documentation	121
10.1.1.1	typedef uint32_t qapi_GPIOINT_Callback_Data_t	121
10.1.1.2	typedef void(* qapi_GPIOINT_CB_t)(qapi_GPIOINT_Callback_Data_t)	121
10.1.1.3	typedef void* qapi_Instance_Handle_t	121
10.1.2	Enumeration Type Documentation	122
10.1.2.1	enum qapi_GPIOINT_Trigger_e	122
10.1.2.2	enumqapi_GPIOINT_Priority_e	122
10.1.3	Function Documentation	122
10.1.3.1	qapi_Status_tqapi_GPIOINT_Register_Interrupt(qapi_Instance_Handle_t pH, uint32_t nGpio, qapi_GPIOINT_CB_t pfnCallback, qapi_G- PRIORITY_Callback_Data_t nData, qapi_GPIOINT_Trigger_e eTrigger, qapi_GPIOINT_Priority_e ePriority, qbool_t bNmi)	122
10.1.3.2	qapi_Status_tqapi_GPIOINT_Deregister_Interrupt(qapi_Instance_Handle_t	123
10.1.3.3	qapi_Status_t qapi_GPIOINT_Set_Trigger (qapi_Instance_Handle_t * pH,	123
10.1.3.4	qapi_Status_t qapi_GPIOINT_Enable_Interrupt (qapi_Instance_Handle_t * pH, uint32_t nGpio)	124
10.1.3.5	qapi_Status_t qapi_GPIOINT_Disable_Interrupt (qapi_Instance_Handle_t * pH, uint32_t nGpio)	124
10.1.3.6	qapi_Status_t qapi_GPIOINT_Trigger_Interrupt (qapi_Instance_Handle_t * pH, uint32_t nGpio)	124
10.1.3.7	qapi_Status_tqapi_GPIOINT_Is_Interrupt_Pending(qapi_Instance_Handle_t pH, uint32_t nGpio, qbool_t * pblsPending)	125
10.2	PMM APIS	125
10.2.1	Data Structure Documentation	126
10.2.1.1	struct qapi_TLMM_Config_t	126
10.2.2	Typedef Documentation	127
10.2.2.1	typedefuint16_tqapi_GPIO_ID_t	127
10.2.3	Enumeration Type Documentation	127
10.2.3.1	enum qapi_GPIO_Direction_t	127
10.2.3.2	enum qapi_GPIO_Pull_t	127
10.2.3.3	enum qapi_GPIO_Drive_t	127
10.2.3.4	enum qapi_GPIO_Value_t	128
10.2.4	Function Documentation	128
10.2.4.1	qapi_Status_t qapi_TLMM_Get_Gpio_ID (qapi_TLMM_Config_t *qapi_TLMM_Config, qapi_GPIO_ID_t * qapi_GPIO_ID)	128
10.2.4.2	qapi_Status_t qapi_TLMM_Release_Gpio_ID (qapi_TLMM_Config_t *qapi_TLMM_Config, qapi_GPIO_ID_t qapi_GPIO_ID)	128
10.2.4.3	qapi_Status_tqapi_TLMM_Config_Gpio(qapi_GPIO_ID_t qapi_GPIO_ID,qapi_TLMM_Config_t * qapi_TLMM_Config)	129
10.2.4.4	qapi_Status_tqapi_TLMM_Drive_Gpio(qapi_GPIO_ID_t qapi_GPIO_ID,uint32_t pin, qapi_GPIO_Value_t value)	129
10.2.4.5	qapi_Status_tqapi_TLMM_Read_Gpio(qapi_GPIO_ID_t qapi_GPIO_ID,uint32_t pin, qapi_GPIO_Value_t * qapi_GPIO_Value)	129
11	STORAGE MODULE	131
11.1	FILE SYSTEM DATA TYPES	132

11.1.1	Data Structure Documentation.....	132
11.1.1.1	struct qapi_FS_Stat_Type_s.....	132
11.1.1.2	structqapi_FS_Statvfs_Type_s.....	132
11.1.1.3	struct qapi_FS_Iter_Entry_s.....	133
11.1.2	Enumeration Type Documentation.....	201
11.1.2.1	enum qapi_FS_Filename_Rule_e.....	201
11.1.2.2	enum qapi_FS_Filename_Encoding_e.....	201
11.2	FILE SYSTEM APIS.....	201
11.2.1	Function Documentation.....	201
11.2.1.1	qapi_FS_Status_t qapi_FS_Open_With_Mode(const char *Path, int Oflag, qapi_FS_Mode_t Mode, int *Fd_ptr).....	201
11.2.1.2	qapi_FS_Status_t qapi_FS_Open(const char *Path, int Oflag, int *Fd_ptr).....	203
11.2.1.3	qapi_FS_Status_t qapi_FS_Read(int Fd, uint8 *Buf, uint32 Count, uint32 *Bytes_Read_Ptr).....	204
11.2.1.4	qapi_FS_Status_t qapi_FS_Write(int Fd, const uint8 *Buf, uint32 Count, uint32 * Bytes_Written_Ptr).....	204
11.2.1.5	qapi_FS_Status_t qapi_FS_Close(int Fd).....	205
11.2.1.6	qapi_FS_Status_t qapi_FS_Rename(const char *Old_Path, const char *New_Path) 205	
11.2.1.7	qapi_FS_Status_t qapi_FS_Truncate(const char *Path, qapi_FS_Offset_t Length).....	205
11.2.1.8	qapi_FS_Status_t qapi_FS_Seek(int Fd, qapi_FS_Offset_t Offset, int Whence, qapi_FS_Offset_t *Actual_Offset_Ptr).....	206
11.2.1.9	qapi_FS_Status_t qapi_FS_Mk_Dir(const char *Path, qapi_FS_Mode_t Mode).....	207
11.2.1.10	qapi_FS_Status_t qapi_FS_Rm_Dir(const char *Path).....	207
11.2.1.11	qapi_FS_Status_t qapi_FS_Unlink(const char *Path).....	208
11.2.1.12	qapi_FS_Status_t qapi_FS_Stat(const char *Path, struct qapi_FS_Stat_Type_s * SBuf).....	208
11.2.1.13	qapi_FS_Status_t qapi_FS_Stat_With_Handle(int Fd, struct qapi_FS_Stat_Type_s * SBuf).....	208
11.2.1.14	qapi_FS_Status_t qapi_FS_Statvfs(const char *Path, struct qapi_FS_ Statvfs_Type_s *SBuf).....	209
11.2.1.15	qapi_FS_Status_t qapi_FS_Iter_Open(const char *Path, qapi_FS_Iter_Handle_t *handle).....	209
11.2.1.16	qapi_FS_Status_t qapi_FS_Iter_Next(qapi_FS_Iter_Handle_t Iter_Hdl, struct qapi_FS_Iter_Entry_s *Iter_Entry).....	209
11.2.1.17	qapi_FS_Status_t qapi_FS_Iter_Close(qapi_FS_Iter_Handle_t Iter_Hdl).....	210
11.2.1.18	qapi_FS_Status_t qapi_FS_Last_Error(void).....	211
11.3	FTL DATA TYPES AND APIS.....	211
11.3.1	Data Structure Documentation.....	211
11.3.2	Typedef Documentation.....	211
11.3.2.1	typedef struct qapi_FTL_info qapi_FTL_info_t.....	211
11.3.2.2	typedef void* qapi_FTL_client_t.....	212
11.3.3	Function Documentation.....	212
11.3.3.1	qapi_Status_t qapi_FTL_Open(qapi_FTL_client_t *handle, const uint8_t *part_name).....	212
11.3.3.2	qapi_Status_t qapi_FTL_Close(qapi_FTL_client_t *handle).....	212
11.3.3.3	qapi_Status_t qapi_FTL_Get_info(qapi_FTL_client_t handle, qapi_FTL_info_t *info).....	213
11.3.3.4	qapi_Status_t qapi_FTL_Read_lpa(qapi_FTL_client_t handle, uint32_t lpa, uint32_t lpa_count, uint8_t *buffer).....	213
11.3.3.5	qapi_Status_t qapi_FTL_Write_lpa(qapi_FTL_client_t handle, uint32_t lpa, uint32_t lpa_count, uint8_t *buffer).....	213
11.3.3.6	qapi_Status_t qapi_FTL_Erase_block(qapi_FTL_client_t handle, uint32_t erase_block, uint32_t erase_block_count).....	214
12	WIRED CONNECTIVITY MODULE.....	216
12.1	USB DATA TYPES.....	216
12.1.1	Data Structure Documentation.....	216
12.1.1.1	union qapi_USB_ioctl_Param.....	216
12.1.2	Typedef Documentation.....	216
12.1.2.1	typedef void(* qapi_USB_App_Rx_Cb_t)(void).....	216
12.1.2.2	typedef union qapi_USB_ioctl_Param qapi_USB_ioctl_Param_t.....	216

12.1.3	Enumeration Type Documentation.....	216
12.1.3.1	enum qapi_USB_ioctl_Cmd_t	216
12.2	USB APIs	217
12.2.1	Function Documentation.....	217
12.2.1.1	qapi_USB_Status_t qapi_USB_Open (void).....	217
12.2.1.2	qapi_USB_Status_t qapi_USB_Read (void**Buffer, uint16 Max_Size).....	217
12.2.1.3	qapi_USB_Status_t qapi_USB_Write (void*Data_Ptr, uint16 Len)	218
12.2.1.4	qapi_USB_Status_t qapi_USB_ioctl (qapi_USB_ioctl_Cmd_t Cmd,qapi_USB_ioctl_Param_t * Param).....	218
13	BUSES MODULE	219
13.1	I2C MASTER APIs	219
13.1.1	Define Documentation	221
13.1.2	Data Structure Documentation.....	221
13.1.2.1	struct qapi_I2CM_Config_t.....	221
13.1.2.2	struct qapi_I2CM_Descriptor_t.....	222
13.1.3	Typedef Documentation	222
13.1.3.1	typedef void(* qapi_I2CM_Transfer_CB_t)(const uint32_t status, void *CB_Parameter)..	222
13.1.4	Enumeration Type Documentation.....	222
13.1.4.1	enum qapi_I2CM_Instance_t	222
13.1.5	Function Documentation.....	223
13.1.5.1	qapi_Status_t qapi_I2CM_Open (qapi_I2CM_Instance_t instance, void **i2c_Handle)	223
13.1.5.2	qapi_Status_t qapi_I2CM_Close (void * i2c_Handle).....	224
13.1.5.3	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)	224
13.1.5.4	qapi_Status_t qapi_I2CM_Power_On (void * i2c_Handle)	225
13.1.5.5	qapi_Status_t qapi_I2CM_Power_Off (void * i2c_Handle)	226
13.2	SPI MASTER APIs.....	227
13.2.1	Data Structure Documentation.....	227
13.2.1.1	struct qapi_SPIM_Config_t.....	227
13.2.1.2	struct qapi_SPIM_Descriptor_t.....	228
13.2.2	Typedef Documentation	228
13.2.2.1	typedef void(* qapi_SPIM_Callback_Fn_t)(uint32_t status, void *callback_Ctxt).....	228
13.2.3	Enumeration Type Documentation.....	253
13.2.3.1	enum qapi_SPIM_Instance_t.....	253
13.2.3.2	enum qapi_SPIM_Shift_Mode_t.....	253
13.2.3.3	enum qapi_SPIM_CS_Polarity_t.....	253
13.2.3.4	enum qapi_SPIM_Byte_Order_t	254
13.2.3.5	enum qapi_SPIM_CS_Mode_t.....	254
13.2.4	Function Documentation.....	254
13.2.4.1	qapi_Status_t qapi_SPIM_Open (qapi_SPIM_Instance_t instance, void ** spi_Handle)	254
13.2.4.2	qapi_Status_t qapi_SPIM_Power_On (void * spi_Handle).....	255
13.2.4.3	qapi_Status_t qapi_SPIM_Power_Off (void * spi_Handle).....	255
13.2.4.4	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)	255
13.2.4.5	qapi_Status_t qapi_SPIM_Close (void * spi_handle).....	256
13.3	UART APIs.....	256
13.3.1	Data Structure Documentation.....	257
13.3.1.1	union QAPI_UART_ioctl_Param	257
13.3.1.2	struct qapi_UART_Open_Config_t.....	257
13.3.2	Typedef Documentation	257
13.3.2.1	typedef union QAPI_UART_ioctl_Param QAPI_UART_ioctl_Param	257
13.3.2.2	typedef void* qapi_UART_Handle_t.....	258
13.3.2.3	typedef void(* qapi_UART_Callback_Fn_t)(uint32_t num_bytes, void *cb_data).....	258
13.3.3	Enumeration Type Documentation.....	258
13.3.3.1	enum qapi_UART_Port_Id_e	258
13.3.3.2	enum qapi_UART_Bits_Per_Char_e.....	258

13.3.3.3	enum	qapi_UART_Num_Stop_Bits_e	259
13.3.3.4	enum	qapi_UART_Parity_Mode_e	259
13.3.3.5	enum	qapi_UART_ioctl_Command_e	259
13.3.3.6	enum	QAPI_Flow_Control_Type	259
13.3.4	Function Documentation		260
13.3.4.1	qapi_Status_t qapi_UART_Close (qapi_UART_Handle_t handle)		260
13.3.4.2	qapi_Status_t qapi_UART_Open (qapi_UART_Handle_t * handle, qapi_UART_Port_Id_e id, qapi_UART_Open_Config_t * config)		260
13.3.4.3	qapi_Status_t qapi_UART_Receive (qapi_UART_Handle_t handle, char *buf, uint32_t buf_Size, void *cb_Data)		260
13.3.4.4	qapi_Status_t qapi_UART_Transmit (qapi_UART_Handle_t handle, char *buf, uint32_t bytes_To_Tx, void *cb_Data)		261
13.3.4.5	qapi_Status_t qapi_UART_Power_On (qapi_UART_Handle_t UART_Handle)		262
13.3.4.6	qapi_Status_t qapi_UART_Power_Off (qapi_UART_Handle_t UART_Handle)		262
13.3.4.7	qapi_Status_t qapi_UART_ioctl (qapi_UART_Handle_t handle, qapi_UART_ - ioctl_Command_e ioctl_Command, void *ioctl_Param)		262
14	LOCATION MODULE		264
14.1	LOCATION APIs		264
14.1.1	Data Structure Documentation		264
14.1.1.1	struct	qapi_Location_Options_t	264
14.1.1.2	struct	qapi_Geofence_Options_t	264
14.1.1.3	struct	qapi_Geofence_Data_t	264
14.1.1.4	struct	qapi_Location_t	265
14.1.1.5	struct	qapi_Location_Callbacks_t	265
14.1.2	Typedef Documentation		265
14.1.2.1	typedef void(* qapi_Capabilities_Callback)(uint16_t capabilitiesMask)		265
14.1.2.2	typedef void(* qapi_Response_Callback)(qapi_Location_Error_t err, uint32_t id)		266
14.1.2.3	typedef void(* qapi_Geofence_Response_Callback)(size_t count, qapi_ - Location_Error_t *err, uint32_t *ids)		266
14.1.2.4	typedef void(* qapi_Tracking_Callback)(qapi_Location_t location)		266
14.1.2.5	typedef void(* qapi_Batching_Callback)(size_t count, qapi_Location_t *location)		266
14.1.2.6	typedef void(* qapi_Geofence_Breach_Callback)(size_t count, uint32_t *ids, qapi_Location_t location, qapi_Geofence_Breach_t breachType, uint64_t timestamp)		267
14.1.3	Enumeration Type Documentation		267
14.1.3.1	enum	qapi_Location_Error_t	267
14.1.3.2	enum	qapi_Location_Flags_t	267
14.1.3.3	enum	qapi_Geofence_Breach_t	268
14.1.3.4	enum	qapi_Geofence_Breach_Mask_t	268
14.1.3.5	enum	qapi_Location_Capabilities_Mask_t	268
14.1.4	Function Documentation		268
14.1.4.1	qapi_Location_Error_t qapi_Loc_Init (qapi_Location_Callbacks_t callbacks)		268
14.1.4.2	qapi_Location_Error_t qapi_Loc_Deinit (void)		269
14.1.4.3	qapi_Location_Error_t qapi_Loc_Start_Tracking (qapi_Location_Options_t *pOptions, uint32_t * pSessionId)		269
14.1.4.4	qapi_Location_Error_t qapi_Loc_Stop_Tracking (uint32_t id)		269
14.1.4.5	qapi_Location_Error_t qapi_Loc_Update_Tracking_Options (uint32_t id, qapi_Location_Options_t * pOptions)		270
14.1.4.6	qapi_Location_Error_t qapi_Loc_Start_Batching (qapi_Location_Options_t pOptions, uint32_t * pSessionId)		270
14.1.4.7	qapi_Location_Error_t qapi_Loc_Stop_Batching (uint32_t id)		271
14.1.4.8	qapi_Location_Error_t qapi_Loc_Update_Batching_Options (uint32_t id, qapi_Location_Options_t * pOptions)		271
14.1.4.9	qapi_Location_Error_t qapi_Loc_Get_Batched_Locations (uint32_t id, size_t count) ...		271
14.1.4.10	qapi_Location_Error_t qapi_Loc_Add_Geofences (size_t count, qapi_ - Geofence_Options_t * pOptions, qapi_Geofence_Data_t * pData, uint32_t ** pIdArray)		272
14.1.4.11	qapi_Location_Error_t qapi_Loc_Remove_Geofences (size_t count, uint32_t * pIDs)		272
14.1.4.12	qapi_Location_Error_t qapi_Loc_Modify_Geofences (size_t count, uint32_t pIDs, qapi_Geofence_Options_t * options)		272
14.1.4.13	qapi_Location_Error_t qapi_Loc_Pause_Geofences (size_t count, uint32_t pIDs) ...		273
14.1.4.14	qapi_Location_Error_t qapi_Loc_Resume_Geofences (size_t count, uint32_t * pIDs)		273

15	TIMER MODULE	274
15.1	TIMER APIS	275
15.1.1	Data Structure Documentation	275
15.1.1.1	struct qapi_TIMER_define_attr_t	275
15.1.1.2	struct qapi_TIMER_set_attr_t	276
15.1.1.3	struct qapi_TIMER_get_info_attr_t	276
15.1.2	Typedef Documentation	276
15.1.2.1	typedef void* qapi_TIMER_handle_t	276
15.1.2.2	typedef void(* qapi_TIMER_cb_t)(uint32_t data)	277
15.1.3	Enumeration Type Documentation	277
15.1.3.1	enum qapi_TIMER_notify_t	277
15.1.3.2	enum qapi_TIMER_unit_type	277
15.1.3.3	enum qapi_TIMER_info_type	277
15.1.4	Function Documentation	278
15.1.4.1	qapi_Status_t qapi_Timer_Def (qapi_TIMER_handle_t * timer_handle, qapi_TIMER_define_attr_t * timer_attr)	278
15.1.4.2	qapi_Status_t qapi_Timer_Set(qapi_TIMER_handle_t timer_handle, qapi_TIMER_set_attr_t * timer_attr)	278
15.1.4.3	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)	278
15.1.4.4	qapi_Status_t qapi_Timer_Sleep(uint64_t timeout, qapi_TIMER_unit_type unit, qbool_t non_deferrable)	279
15.1.4.5	qapi_Status_t qapi_Timer_Undef(qapi_TIMER_handle_t timer_handle)	279
15.1.4.6	qapi_Status_t qapi_Timer_Stop(qapi_TIMER_handle_t timer_handle)	279
15.1.4.7	qapi_Status_t qapi_Timer_set_absolute(qapi_TIMER_handle_t timer, uint64_t abs_time) 264	
15.2	PMIC RTC APIS	264
15.2.1	Data Structure Documentation	264
15.2.2	Typedef Documentation	265
15.2.3	Enumeration Type Documentation	265
15.2.3.1	enum qapi_PM_Rtc_Cmd_Type_t	265
15.2.3.2	enum qapi_PM_Rtc_Display_Type_t	265
15.2.3.3	enum qapi_PM_Rtc_Alarm_Type_t	265
15.2.4	Function Documentation	265
15.2.4.1	qapi_Status_t qapi_PM_Rtc_Init(void)	265
15.2.4.2	qapi_Status_t qapi_PM_Set_Rtc_Display_Mode(qapi_PM_Rtc_Display_Type_t mode) 266	
15.2.4.3	qapi_Status_t qapi_PM_Rtc_Read_Cmd (qapi_PM_Rtc_Julian_Type_t * qapi_current_time_ptr)	266
15.2.4.4	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)	267
16	HARDWARE ENGINE APIS	268
16.1	ADC DATA TYPES	269
16.1.1	Define Documentation	269
16.1.2	Data Structure Documentation	269
16.1.2.1	struct qapi_ADC_Read_Result_t	269
16.1.2.2	struct qapi_Adc_Input_Properties_Type_t	270
16.1.2.3	struct qapi_AdcTM_Input_Properties_Type_t	270
16.1.2.4	struct qapi_ADC_Range_t	270
16.1.2.5	struct qapi_ADC_Threshold_Result_t	270
16.1.2.6	struct qapi_ADC_Device_Properties_t	271
16.1.2.7	struct qapi_AdcTM_Callback_Payload_Type_t	271
16.1.2.8	struct qapi_AdcTM_Range_Type_t	271
16.1.2.9	struct qapi_AdcTM_Request_Params_Type_t	271
16.1.3	Typedef Documentation	272
16.1.3.1	typedef void(* qapi_ADC_Threshold_CB_t)(void *ctxt, const qapi_ADC_Threshold_Result_t *result)	272
16.1.3.2	typedef void(* qapi_AdcTM_Threshold_Cb_Type)(void *ctxt, const qapi_ADC_Threshold_Result_t *result)	272
16.1.4	Enumeration Type Documentation	272

16.1.4.1	enum qapi_ADC_Amp_Threshold_t	272
16.2	ADC APIs	272
16.2.1	Function Documentation	274
16.2.1.1	qapi_Status_t qapi_ADC_Open (qapi_ADC_Handle_t * Handle, uint32_t Attributes) 274	
16.2.1.2	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)	275
16.2.1.3	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)	275
16.2.1.4	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)	276
16.2.1.5	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)	276
16.2.1.6	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)	276
16.2.1.7	qapi_Status_t qapi_ADC_TM_Enable_Thresholds(qapi_ADC_Handle_t Handle, uint32_t Enable, qapi_ADC_Amp_Threshold_t Threshold_Type)	277
16.2.1.8	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)	278
16.2.1.9	qapi_Status_t qapi_ADC_Close (qapi_ADC_Handle_t Handle, qbool_t keep_enabled) 278	
16.3	TSENS DATA TYPES	279
16.3.1	Data Structure Documentation	279
16.3.1.1	struct qapi_TSENS_CallbackPayloadType_t	279
16.3.1.2	struct qapi_TSENS_Result_t	279
16.3.2	Typedef Documentation	279
16.3.2.1	typedef void(* QAPI_Tsens_Threshold_Cb_Type)(void *pCtxt, const qapi_TSENS_CallbackPayloadType_t *pPayload)	279
	typedef void* qapi_TSENS_Handle_t	280
16.3.3	Enumeration Type Documentation	280
16.3.3.1	enum qapi_TSENS_ThresholdType_t	280
16.4	TSENS APIs	280
16.4.1	Function Documentation	281
16.4.1.1	qapi_Status_t qapi_TSENS_Open (qapi_TSENS_Handle_t * Handle)	281
16.4.1.2	qapi_Status_t qapi_TSENS_Get_Num_Sensors(qapi_TSENS_Handle_t Handle, int32_t * Num_Sensors_Ptr)	281
16.4.1.3	qapi_Status_t qapi_TSENS_Get_Temp(qapi_TSENS_Handle_t Handle, uint32_t Sensor_Num, qapi_TSENS_Result_t * Temp_Result_Ptr)	282
16.4.1.4	qapi_Status_t qapi_TSENS_Get_Calibration_Status(qapi_TSENS_Handle_t Handle, uint32_t Sensor_Num)	282
16.4.1.5	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)	283
16.4.1.6	qapi_Status_t qapi_TSENS_Set_Enable_Thresholds(qapi_TSENS_Handle_t Handle, int32_t Enable_Threshold)	283
16.4.1.7	qapi_Status_t qapi_TSENS_Close(qapi_TSENS_Handle_t Handle)	284
17	ALI IOT CLOUD	285
17.1	IOT DEBUG DATA TYPES	285
17.1.1	Enumeration Type Documentation	285
17.1.1.1	enum IOT_LogLevel	285
17.2	IOT DEBUG APIs	285
17.2.1	Function Documentation	285
17.2.1.1	void IOT_OpenLog(const char *ident)	285
17.2.1.2	void IOT_CloseLog(void)	285
17.2.1.3	void IOT_SetLogLevel(IOT_LogLevel level)	285

17.2.1.4	void IOT_DumpMemoryStats(IOT_LogLevel level).....	286
17.3	IOT MQTT DATA TYPES.....	286
17.3.1	Enumeration Type Documentation.....	286
17.3.1.1	enum iotx_mqtt_qos_t.....	286
17.3.1.2	enum iotx_mqtt_event_type_t.....	286
17.3.2	Typedef Documentation.....	287
17.3.2.1	typedef void (*iotx_mqtt_event_handle_func_fpt)(void *pcontext, void *pclient, iotx_mqtt_event_msg_pt msg);.....	287
17.3.3	Data Structure Documentation.....	287
17.3.3.1	struct iotx_mqtt_topic_info_t.....	287
17.3.3.2	struct iotx_mqtt_event_msg_t.....	287
17.3.3.3	struct iotx_mqtt_event_handle_t.....	288
17.3.3.4	struct iotx_mqtt_param_t.....	288
17.4	IOT MQTT APIs.....	289
17.4.1	Function Documentation.....	289
17.4.1.1	int IOT_SetupConnInfo(const char *product_key,const char *device_name,const char *device_secret,void **info_ptr);.....	289
17.4.1.2	void *IOT_MQTT_Construct(iotx_mqtt_param_t *pInitParams);.....	289
17.4.1.3	int IOT_MQTT_Destroy(void **phandle);.....	289
17.4.1.4	int IOT_MQTT_Yield(void *handle, int timeout_ms);.....	290
17.4.1.5	int IOT_MQTT_CheckStateNormal(void *handle);.....	290
17.4.1.6	int IOT_MQTT_Subscribe(void *handle,const char *topic_filter,iotx_mqtt_qos_t qos,iotx_mqtt_event_handle_func_fpt topic_handle_func,void *pcontext);.....	290
17.4.1.7	int IOT_MQTT_Unsubscribe(void *handle, const char *topic_filter);.....	290
17.4.1.8	int IOT_MQTT_Publish(void *handle, const char *topic_name, iotx_mqtt_topic_info_pt topic_msg);.....	291
17.5	IOT COAP DATA TYPES.....	291
17.5.1	Enumeration Type Documentation.....	291
17.5.1.1	enum iotx_ret_code_t.....	291
17.5.1.2	enum otx_content_type_t.....	291
17.5.1.3	enum iotx_msg_type_t.....	291
17.5.1.4	enum iotx_coap_event_t.....	291
17.5.1.5	enum iotx_coap_resp_code_t.....	291
17.5.2	Typedef Documentation.....	292
17.5.2.1	typedef void (*iotx_event_handle_t)(void *context, iotx_coap_event_t event, void *p_data); 292	
17.5.2.2	typedef void (*iotx_response_callback_t)(void *p_arg, void *p_message);.....	292
17.5.2.3	struct otx_deviceinfo_t.....	292
17.5.2.4	struct iotx_coap_config_t.....	292
17.5.2.5	struct iotx_message_t.....	292
17.6	IOT COAP APIs.....	293
17.6.1	Function Documentation.....	293
17.6.1.1	iotx_coap_context_t *IOT_CoAP_Init(iotx_coap_config_t *p_config);.....	293
17.6.1.2	void IOT_CoAP_Deinit(iotx_coap_context_t **p_context);.....	293
17.6.1.3	int IOT_CoAP_DeviceNameAuth(iotx_coap_context_t *p_context);.....	293
17.6.1.4	int IOT_CoAP_Yield(iotx_coap_context_t *p_context);.....	293
17.6.1.5	int IOT_CoAP_SendMessage(iotx_coap_context_t *p_context, char *p_path, iotx_message_t *p_message);.....	293
17.6.1.6	int IOT_CoAP_GetMessagePayload(void *p_message, unsigned char **pp_payload, int *p_len); 294	
17.6.1.7	int IOT_CoAP_GetMessageCode(void *p_message, iotx_coap_resp_code_t *p_resp_code); 294	
18	AT ENGINE.....	295
18.1	AT ENGINE APIs.....	295
18.1.1	Function Documentation.....	295
18.1.1.1	unsigned short qapi_DAM_Visual_AT_Input (const unsigned char *data, unsigned short length); 295	
18.1.1.2	void qapi_DAM_Visual_AT_Output (unsigned char *data, unsigned short length);.....	295
18.1.1.3	void qapi_DAM_Update_App (void).....	295
19	DEVICE INFORMATION MODULE	296

19.1	DEVICE INFORMATION	296
19.1.1	Define Documentation	296
19.1.1.1	#define QAPI_DEVICE_INFO_BUF_SIZE 128	296
19.1.2	Data Structure Documentation	296
19.1.2.1	struct qapi_Device_Info_t	296
19.1.2.2	union qapi_Device_Info_t.u	296
19.1.2.3	struct qapi_Device_Info_t.u.valuebuf	296
19.1.3	Enumeration Type Documentation	297
19.1.3.1	enum qapi_Device_Info_ID_t	297
19.1.3.2	enum qapi_Device_Info_Type_t	297
19.1.4	Function Documentation	297
19.1.4.1	qapi_Status_t qapi_Device_Info_Init(void)	297
19.1.4.2	qapi_Status_t qapi_Device_Info_Get(qapi_Device_Info_ID_t id, qapi_Device_Info_t* info)	298
19.1.4.3	qapi_Status_t qapi_Device_Info_Set_Callback(qapi_Device_Info_ID_t id, qapi_Device_Info_Callback_t callback)	298
19.1.4.4	qapi_Status_t qapi_Device_Info_Release(void)	298
19.1.4.5	qapi_Status_t qapi_Device_Info_Reset(void)	299
19.2	QAPI_NET_DATATYPES	299
20	USE CASES	300
20.1	DSS API USE CASE	300
20.2	SOCKET API USE CASES	300
20.2.1	Server Socket	300
20.2.2	Client Socket	301
20.3	TLS/DTLS API USE CASES	302
20.3.1	TLS/DTLS Context Object Creation	302
20.3.2	TLS/DTLS Certificate or CA List, or PSK Table Store and Load to SSL Context	302
20.3.3	TLS/DTLS Connection Object Creation	303
20.3.4	TLS/DTLS Configuration of a Connection Object	303
20.3.5	Secure Socket Data Transfer over a TLS/DTLS Connection	304
20.3.6	Close an SSL Connection TLS/DTLS Connection and Socket	305
20.3.7	TLS/DTLS Close Context Object	306
A	REFERENCES	307
	Acronyms and Terms	307

Version History

Date	Version	Description	Author
2017-11-27	V1.00	Original	Wenjie.lai

1 Introduction

1.1 Purpose

This document is the reference specification for the Qualcomm Application Programming Interface (QAPI) for the SIM7000 Series 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`.

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.

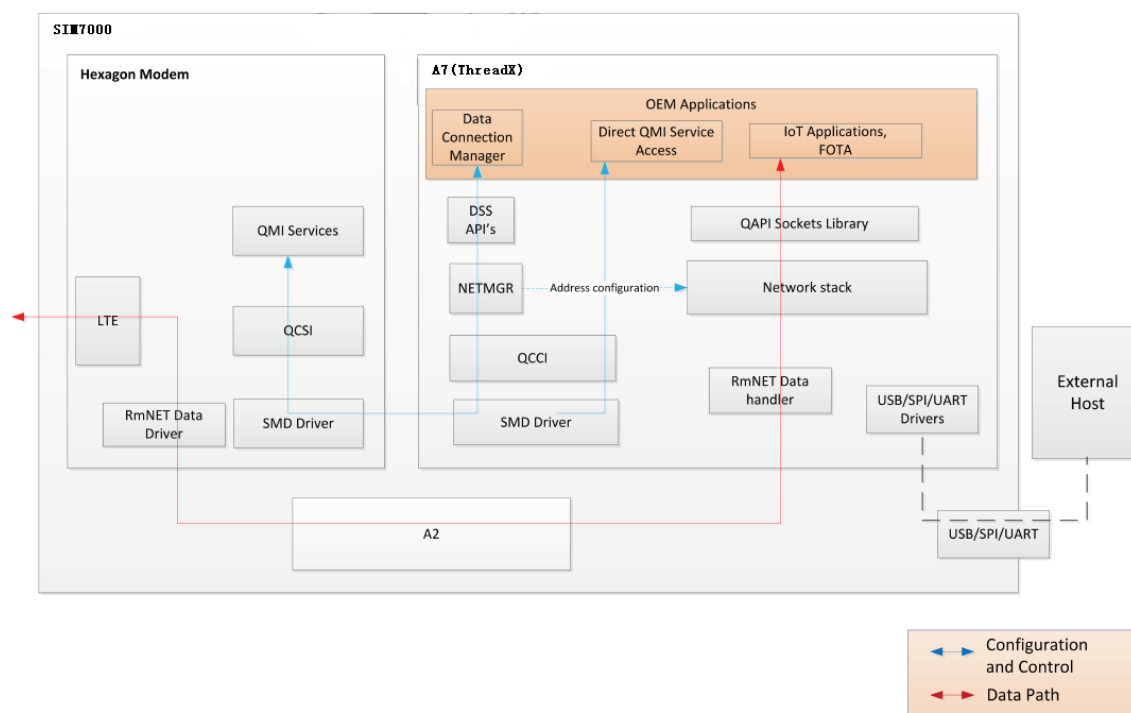


Figure 2-1 Data call architecture in ThreadX OS

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 qapi_DSS_Net_Ev_CB_t

```
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.

<i>QAPI_DSS_DATA_BEARER_TECH_FMC_E</i>	Fixed	mobile	convergence.
<i>QAPI_DSS_DATA_BEARER_TECH_HRPD_E</i>			HRPD.
<i>QAPI_DSS_DATA_BEARER_TECH_3GPP2_WLAN_E</i>			IWLAN.
<i>QAPI_DSS_DATA_BEARER_TECH_WCDMA_E</i>			WCDMA.
<i>QAPI_DSS_DATA_BEARER_TECH_GPRS_E</i>			GPRS.
<i>QAPI_DSS_DATA_BEARER_TECH_HSDPA_E</i>			HSDPA.
<i>QAPI_DSS_DATA_BEARER_TECH_HSUPA_E</i>			HSUPA.
<i>QAPI_DSS_DATA_BEARER_TECH_EDGE_E</i>			EDGE.
<i>QAPI_DSS_DATA_BEARER_TECH_LTE_E</i>			LTE.
<i>QAPI_DSS_DATA_BEARER_TECH_HSDPA_PLUS_E</i>			HSDPA+.
<i>QAPI_DSS_DATA_BEARER_TECH_DC_HSDPA_PLUS_E</i>		DC	HSDPA+.
<i>QAPI_DSS_DATA_BEARER_TECH_HSPA_E</i>			HSPA.
<i>QAPI_DSS_DATA_BEARER_TECH_64_QAM_E</i>		64	QAM.
<i>QAPI_DSS_DATA_BEARER_TECH_TDSCDMA_E</i>			TD-SCDMA.
<i>QAPI_DSS_DATA_BEARER_TECH_GSM_E</i>			GSM.
<i>QAPI_DSS_DATA_BEARER_TECH_3GPP_WLAN_E</i>	IWLAN.		

3.1.4.7 enum qapi_DSS_Call_Tech_Type_e

Call technology.

Enumerator:

<i>QAPI_DSS_CALL_TECH_INVALID_E</i>	Invalid technology.
<i>QAPI_DSS_CALL_TECH_CDMA_E</i>	CDMA.
<i>QAPI_DSS_CALL_TECH_UMTS_E</i>	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

in	<i>mode</i>	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

in	<i>mode</i>	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_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	<i>user_cb_fn</i>	Client callback function used to post event indications.
in	<i>user_data</i>	Pointer to the client context block (cookie). The value may be NULL.
in	<i>hndl</i>	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 qapi_DSS_Get_Data_Srvc_Hndl() .
-----------	-------------	--

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_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 qapi_DSS_Get_Data_Srvc_Hndl() .
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_Srv_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_Srv_Hndl\(\)](#) indicating the data call bring-up status.

Parameters

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

Returns

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

Dependencies

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

A valid handle must be obtained by [qapi_DSS_Get_Data_Srv_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 qapi_DSS_Get_Data_Srvc_Hndl() .
-----------	-------------	--

Returns

- QAPI_OK – Data call stop request was sent successfully.
- QAPI_ERROR – Data call stop request was unsuccessful.

Dependencies

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

A valid handle must be obtained by [qapi_DSS_Get_Data_Srvc_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 qapi_DSS_Get_Data_Srvc_Hndl() .
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_Srvc_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 qapi_DSS_Get_Data_Srvc_Hndl() .
-----------	-------------	--

Returns

QAPI_OK – Packet data transfer statistics were reset successfully.

QAPI_ERROR – Packet data transfer statistics reset was unsuccessful.

Dependencies

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

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

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_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 qapi_DSS_Get_Data_Srvc_Hndl() .
out	<i>ce_reason</i>	Bufferto 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_Srvc_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_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 qapi_DSS_Get_Data_Srvc_Hndl() .
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_Srvc_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_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 qapi_DSS_Get_Data_Srvc_Hndl() .
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 qapi_DSS_Get_Data_Srvc_Hndl() .
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_Srvc_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 qapi_DSS_Get_Data_Srvc_Hndl() .
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 qapi_DSS_Get_Data_Srv_Hndl() .
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_Srv_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 qapi_DSS_Get_Data_Srv_Hndl() .
-----------	-------------	---

out	<i>info_ptr</i>	Buffer containing the IP address information.
in	<i>len</i>	Number of IP address buffers

Returns

- QAPI_OK – IP address query was successful.
- QAPI_ERROR – IP address query was unsuccessful.

Dependencies

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

A valid handle must be obtained by [qapi_DSS_Get_Data_Srv_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_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 qapi_DSS_Get_Data_Srv_Hndl() .
out	<i>info_ptr</i>	Buffer containing the IP address information.
in	<i>addr_family</i>	IPv4 / IPv6

Returns

- QAPI_OK – IP address query was successful.
- QAPI_ERROR – IP address query was unsuccessful.

Dependencies

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

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

4.1 QAPI Socket Macros and Data Structures

This section provides the QAPI socket macros and data structures.

BSD Socket Error Codes

- `#define ENOBUFS 1`
- `#define ETIMEDOUT 2`
- `#define EISCONN 3`
- `#define EOPNOTSUPP 4`
- `#define ECONNABORTED 5`
- `#define EWOULDBLOCK 6`
- `#define ECONNREFUSED 7`
- `#define ECONNRESET 8`
- `#define 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_TCPSACK 0x00200
- #define SO_WINSIZE 0x00400
- #define SO_TIMESTAMP 0x00800
- #define SO_BIGCWND 0x01000
- #define SO_HDRINCL 0x02000
- #define SO_NOSLOWSTART 0x04000
- #define SO_FULLMSS 0x08000
- #define SO_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

#define AF_UNSPEC 0

Address family unspecified.

#define AF_INET 2

Address family IPv4.

#define AF_INET6 3

Address family IPv6.

#define AF_INET_DUAL 464

Address family IPv4 and IPv6.

#define SOCK_STREAM 1

Socket stream (TCP).

#define SOCK_DGRAM 2

Socket datagram (UDP).

#define SOCK_RAW 3

Raw socket.

#define ENOBUFS 1

No buffer space is available.

#define ETIMEDOUT 2

Operation timed out.

#define EISCONN 3

Socket is already connected.

#define EOPNOTSUPP 4

Operation is not supported.

#define ECONNABORTED 5

Software caused a connection abort.

#define EWOULDBLOCK 6

Socket is marked nonblocking and the requested operation will block.

#define ECONNREFUSED 7

Connection was refused.

#define ECONNRESET 8

Connection was reset by peer.

#define ENOTCONN 9

Socket is not connected.

#define EBADF 9

An invalid descriptor was specified.

#define EALREADY 10

Operation is already in progress.

#define EINVAL 11

Invalid argument was passed.

#define EMSGSIZE 12

Message is too long.

#define EPIPE 13

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

#define EDESTADDRREQ 14

Destination address is required.

#define ESHUTDOWN 15

Cannot send after a socket shutdown.

#define ENOPROTOOPT 16

Protocol is not available.

#define EHAVEOOB 17

Out of band.

#define ENOMEM 18

No memory is available.

#define EADDRNOTAVAIL 19

Cannot assign the requested address.

#define EADDRINUSE 20

Address is already in use.

#define EAFNOSUPPORT 21

Address family is not supported by the protocol family.

#define EINPROGRESS 22

Operation is in progress.

#define ELOWER 23

Lower layer (IP) error.

#define ENOTSOCK 24

Socket operation on nonsocket.

#define EIEIO 27

I/O error.

#define ETOOMANYREFS 28

Too many references.

#define EFAULT 29

Bad address.

#define ENETUNREACH 30

Network is unreachable.

#define SOL_SOCKET -1

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

#define SOL_SOCKET -1

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

#define SO_ACCEPTCONN 0x00002

Socket has had listen().

#define SO_REUSEADDR 0x00004

Allow local address reuse.

#define SO_KEEPALIVE 0x00008

Keep connections alive.

#define SO_DONTROUTE 0x00010

Not used.

#define SO_BROADCAST 0x00020

Not used.

#defineSO_USELOOPBACK0x00040

Not used.

#defineSO_LINGER0x00080

Linger on close if data is present.

#defineSO_OOBLINELINE0x00100

Leave the received OOB data in line.

#defineSO_TCPSACK0x00200

Allow TCP SACK (selective acknowledgment).

#defineSO_WINSCALE0x00400

Set the scaling window option.

#defineSO_TIMESTAMP0x00800

Set the TCP timestamp option.

#defineSO_BIGCWND0x01000

Large initial TCP congestion window.

#defineSO_HDRINCL0x02000

User access to IP header for SOCK_RAW.

#defineSO_NOSLOWSTART0x04000

Suppress slowstart on this socket.

#defineSO_FULLMSS0x08000

Not used.

#defineSO_SNDTIMEO0x1005

Send a timeout.

#define SO_RCVTIMEO 0x1006

Receive a timeout.

#define SO_ERROR 0x1007

Socket error.

#define SO_RXDATA 0x1011

Get a count of bytes in sb_rcv.

#define SO_TXDATA 0x1012

Get a count of bytes in sb_snd.

#define SO_MYADDR 0x1013

Return my IP address.

#define SO_NBLOCK 0x1014

Set socket to Nonblocking mode.

#define SO_BLOCK 0x1015

Set socket to Blocking mode.

#define SO_NONBLOCK 0x1016

Set/get blocking mode via the optval parameter.

#define SO_CALLBACK 0x1017

Set/get the TCP zero_copy callback routine.

#define SO_UDPCALLBACK 0x1019

Set/get the UDP zero_copy callback routine.

#define IPPROTO_IP 0

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

#define IP_HDRINCL 2

IP header is included with the data.

#define IP_MULTICAST_IF 9

Set/get the IP multicast interface.

#define IP_MULTICAST_TTL 10

Set/get the IP multicast TTL.

#define IP_MULTICAST_LOOP 11

Set/get the IP multicast loopback.

#define IP_ADD_MEMBERSHIP 12

Add an IPv4 group membership.

#define IP_DROP_MEMBERSHIP 13

Drop an IPv4 group membership.

#define IPV6_MULTICAST_IF 80

Set the egress interface for multicast traffic.

#define IPV6_MULTICAST_HOPS 81

Set the number of hops.

#define IPV6_MULTICAST_LOOP 82

Enable/disable loopback for multicast.

#define IPV6_JOIN_GROUP 83

Join an IPv6 MC group.

#define IPV6_LEAVE_GROUP 84

Leave an IPv6 MC group.

#define IP_OPTIONS 1

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

#define IP_TOS 3

IPv4 type of service and precedence.

#define IP_TTL_OPT 4

IPv4 time to live.

#define IPV6_SCOPEID 14

IPv6 IF scope ID.

#define IPV6_UNICAST_HOPS 15

IPv6 hop limit.

#define IPV6_TCLASS 16

IPv6 traffic class.

#define MSG_OOB 0x1

Send/receive out-of-band data.

#define MSG_PEEK 0x2

Peek at the incoming message.

#define MSG_DONTROUTE 0x4

Send without using routing tables.

#define MSG_DONTWAIT 0x20

Send/receive is nonblocking.

#define MSG_ZEROCOPYSEND 0x1000

Send with zero-copy.

#define QAPI_NET_WAIT_FOREVER (0xFFFFFFFF)

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

```
#define FD_ZERO( set ) qapi_fd_zero((set))
```

Clears a set.

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

Removes a given file descriptor from a set.

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

Adds a given file descriptor from a set.

```
#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 structin_addr

IPv4 Internet address.

Data fields

Type	Parameter	Description
uint32_t	s_addr	IPv4 address in network order.

4.1.2.2 structsockaddr_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 structin6_addr

IPv6 Internet address.

Data fields

Type	Parameter	Description
uint8_t	s_addr	128-bit IPv6 address.

4.1.2.4 structip46addr_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 unionip46addr_n.a

Data fields

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

4.1.2.6 unionip46addr_n.g

Data fields

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

4.1.2.7 structsockaddr_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 structip46addr

Socket IPv4/IPv6 Internet address union.

Data fields

Type	Parameter	Description
uint16_t	type	AF_INET or AF_INET6.

union ip46addr	a	Address union.
--------------------------------	---	----------------

4.1.2.9 unionip46addr.a

Data fields

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

4.1.2.10 structsockaddr

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 structfd_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"> • AF_INET – IPv4 Internet protocols • AF_INET6 – IPv6 Internet protocols
in	<i>type</i>	Transport mechanism used for communication. The supported types are: <ul style="list-style-type: none"> • SOCK_STREAM – TCP • SOCK_DGRAM – UDP
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 qapi_socket() .
in	<i>addr</i>	Pointer to an address to be assigned to the socket. The actual address structure passed for the <i>addr</i> argument will depend on the address family.
in	<i>addrlen</i>	Specifies the size, in bytes, of the address pointed to by <i>addr</i> .

Returns

On success, 0 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 qapi_socket()) that refers to a SOCK_STREAM socket.
in	<i>backlog</i>	Define the maximum length to which the queue of pending connections for the handle may grow.

Returns

On success, 0 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 <code>qapi_socket()</code> .
-----------	---------------	--

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

Returns

On success, 0 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 qapi_socket() .
in	<i>level</i>	Protocol level at which the option exists.
in	<i>optname</i>	Name of the option.
in	<i>optval</i>	Pointer to a buffer in which the value for the requested option is to be returned.
in	<i>optlen</i>	Option length in bytes.

Returns

On success, 0 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 qapi_socket() .
-----------	---------------	---

Returns

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

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 qapi_socket() .
-----------	---------------	---

Returns

Socket error code or ENOTSOCK if socket is not found

4.11 Receive a Message from a Socket

4.11.1 Function Documentation

4.11.1.1 `int qapi_recvfrom`

```
int qapi_recvfrom ( 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 qapi_socket() .
in	<i>buf</i>	Pointer to a buffer for the received message.
in	<i>len</i>	Number of bytes to receive.
in	<i>flags</i>	0, or it is formed by ORing one or more of: <ul style="list-style-type: none"> • MSG_PEEK – Causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data. • MSG_OOB – Requests receipt of out-of-band data that would not be received in the normal data stream. • MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.
in	<i>from</i>	If not NULL, and the underlying protocol provides the source address, this source address is filled in. When NULL, nothing is filled in; in this case, fromlen is not used, and should also be NULL.
in	<i>fromlen</i>	This is a value-result argument, which the caller should initialize before the call to the size of the buffer associated with from, and modified on return to indicate the actual size of the source address.

Returns

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

4.12 Receive a Message from a Connected Socket

4.12.1 Function Documentation

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

Receives a message from a socket.

The [qapi_recv\(\)](#) call is normally used only on a connected socket and is identical to `qapi_recvfrom(handle, buf, len, flags, NULL, NULL)`

Parameters

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

in	<i>flags</i>	0, or it is formed by ORing one or more of: <ul style="list-style-type: none"> MSG_PEEK – Causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data. MSG_OOB – Requests receipt of out-of-band data that would not be received in the normal data stream. MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.
-----------	--------------	---

Returns

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

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 qapi_socket() .
in	<i>buf</i>	Pointer to a buffer containing the message to be sent.
in	<i>len</i>	Number of bytes to send.
in	<i>flags</i>	0, or it is formed by ORing one or more of: <ul style="list-style-type: none"> MSG_OOB – Sends out-of-band data on sockets that support this notion (e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data. MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK. MSG_DONTROUTE – Don not use a gateway to send the packet; only send it to hosts on directly-connected networks. This is usually used only by diagnostic or routing programs.
in	<i>to</i>	Pointer to the address of the target.
in	<i>tolen</i>	Size in bytes of the target address.

Returns

The number of bytes sent, or -1 if an error occurred and 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 qapi_socket() .
in	<i>buf</i>	Pointer to a buffer containing message to be sent.
in	<i>len</i>	Number of bytes to send.
in	<i>flags</i>	0, or it is formed by ORing one or more of: <ul style="list-style-type: none"> MSG_OOB – Sends out-of-band data on sockets that support this notion (e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data. MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK. MSG_DONTROUTE – Do not use a gateway to send the packet; only send it to hosts on directly-connected networks. This is usually used only by diagnostic or routing programs.

Returns

The number of bytes sent, or -1 if an error occurred and `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

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

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

Returns

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

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 qapi_socket() .
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 qapi_socket() .
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

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

Returns

On success, 0 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 qapi_socket()
in	<i>addr</i>	Pointer to a user buffer of <code>sockaraddr</code> type which is filled by the API with the peer <code>addr</code> info.
in	<i>addrlen</i>	Specifies the size, in bytes, of the address pointed to by <code>addr</code>

Returns

On success, 0 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

#define QAPI_NET_SSL_MAX_CERT_NAME_LEN (32)

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

#define QAPI_NET_SSL_MAX_NUM_CERTS (10)

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

#define QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH 8

Maximum number of cipher suites that can be configured.

#define QAPI_NET_SSL_INVALID_HANDLE (0)

Invalid handle.

#define QAPI_NET_SSL_PROTOCOL_UNKNOWN 0x00

Unknown SSL protocol version.

#define QAPI_NET_SSL_PROTOCOL_TLS_1_0 0x31

TLS version 1.0.

#define QAPI_NET_SSL_PROTOCOL_TLS_1_1 0x32

TLS version 1.1.

#define QAPI_NET_SSL_PROTOCOL_TLS_1_2 0x33

TLS version 1.2.

#define QAPI_NET_SSL_PROTOCOL_DTLS_1_0 0xEF

DTLS version 1.0.

#define QAPI_NET_SSL_PROTOCOL_DTLS_1_2 0xED

DTLS version 1.2.

#define QAPI_NET_TLS_PSK_WITH_RC4_128_SHA 0x008A

TLS PSK with RC4 128 SHA.

#define QAPI_NET_TLS_PSK_WITH_3DES_EDE_CBC_SHA 0x008B

TLS PSK with 3DES EDE CBC SHA.

#define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA 0x008C

TLS PSK with AES 128 CBC SHA.

#define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA 0x008D

TLS PSK with AES 256 CBC SHA.

#define QAPI_NET_TLS_PSK_WITH_AES_128_GCM_SHA256 0x00A8

TLS PSK with AES_128 GCM SHA256.

#define QAPI_NET_TLS_PSK_WITH_AES_256_GCM_SHA384 0x00A9

TLS PSK with AES 256 GCM SHA384.

#define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA256 0x00AE

TLS PSK with AES 128 CBC SHA256.

#define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA384 0x00AF

TLS PSK with AES 256 CBC SHA384.

#define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA 0x002F

Cipher TLS RSA with AES 128 CBC SHA.

#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA 0x0033

Cipher TLS DHE RSA with AES 128 CBC SHA.

#define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA 0x0035

Cipher TLS RSA with AES 256 CBC SHA.

#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA 0x0039

Cipher TLS DHE RSA with AES 256 CBC SHA.

#define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA256 0x003C

Cipher TLS RSA with AES 128 CBC SHA256.

```
#define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA256 0x003D
```

Cipher TLS RSA with AES 256 CBC SHA256.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 0x0067
```

Cipher TLS DHE RSA with AES 128 CBC SHA256.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 0x006B
```

Cipher TLS DHE RSA with AES 256 CBC SHA256.

```
#define QAPI_NET_TLS_RSA_WITH_AES_128_GCM_SHA256 0x009C
```

Cipher TLS RSA with AES 128 GCM SHA256.

```
#define QAPI_NET_TLS_RSA_WITH_AES_256_GCM_SHA384 0x009D
```

Cipher TLS RSA with AES 256 GCM SHA384.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 0x009E
```

Cipher TLS DHE RSA with AES 128 GCM SHA256.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 0x009F
```

Cipher TLS DHE RSA with AES 256 GCM SHA384.

```
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA 0xC004
```

Cipher TLS ECDH ECDSA with AES 128 CBC SHA.

```
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA 0xC005
```

Cipher TLS ECDH ECDSA with AES 256 CBC SHA.

```
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA 0xC009
```

Cipher TLS ECDHE ECDSA with AES 128 CBC SHA.

```
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA 0xC00A
```

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA.

```
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA 0xC00E
```

Cipher TLS ECDH RSA with AES 128 CBC SHA.

```
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA 0xC00F
```

Cipher TLS ECDH RSA with AES 256 CBC SHA.

```
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA 0xC013
```

Cipher TLS ECDHE RSA with AES 128 CBC SHA.

```
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA 0xC014
```

Cipher TLS ECDHE RSA with AES 256 CBC SHA.

```
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256  
0x- C023
```

Cipher TLS ECDHE ECDSA with AES 128 CBC SHA256.

```
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384  
0x- C024
```

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA384.

```
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256  
0x- C025
```

Cipher TLS ECDH ECDSA with AES 128 CBC SHA256.

```
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384  
0x- C026
```

Cipher TLS ECDH ECDSA with AES 256 CBC SHA384.

```
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256  
0xC027
```

Cipher TLS ECDHE RSA with AES 128 CBC SHA256.

```
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384  
0xC028
```

Cipher TLS ECDHE RSA with AES 256 CBC SHA384.

```
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 0xC029
```

Cipher TLS ECDH RSA with AES 128 CBC SHA256.

```
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 0xC02A
```

Cipher TLS ECDH RSA with AES 256 CBC SHA384.

```
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256  
0x- C02B
```

Cipher TLS ECDHE ECDSA with AES 128 GCM SHA256.

```
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384  
0x- C02C
```

Cipher TLS ECDHE ECDSA with AES 256 GCM SHA384.

```
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256  
0x- C02D
```

Cipher TLS ECDH ECDSA with AES 128 GCM SHA256.

```
#define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384  
0x- C02E
```

Cipher TLS ECDH ECDSA with AES 256 GCM SHA384.

```
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256  
0xC02- F
```

Cipher TLS ECDHE RSA with AES 128 GCM SHA256.

```
#define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384  
0xC030
```

Cipher TLS ECDHE RSA with AES 256 GCM SHA384.

```
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 0xC031
```

Cipher TLS ECDH RSA with AES 128 GCM SHA256.

```
#define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 0xC032
```

Cipher TLS ECDH RSA with AES 256 GCM SHA384.

```
#define QAPI_NET_TLS_RSA_WITH_AES_128_CCM 0xC09C
```

Cipher TLS RSA with AES 128 CCM.

```
#define QAPI_NET_TLS_RSA_WITH_AES_256_CCM 0xC09D
```

Cipher TLS RSA with AES 256 CCM.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM 0xC09E
```

Cipher TLS DHE RSA with AES 128 CCM.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM 0xC09F
```

Cipher TLS DHE RSA with AES 256 CCM.

```
#define QAPI_NET_TLS_RSA_WITH_AES_128_CCM_8 0xC0A0
```

Cipher TLS RSA with AES 128 CCM 8.

```
#define QAPI_NET_TLS_RSA_WITH_AES_256_CCM_8 0xC0A1
```

Cipher TLS RSA with AES 256 CCM 8.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM_8 0xC0A2
```

Cipher TLS DHE RSA with AES 128 CCM 8.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM_8 0xC0A3
```

Cipher TLS DHE RSA with AES 256 CCM 8.

```
#define QAPI_NET_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC13
```

Cipher TLS ECDHE RSA with CHACHA20 POLY1305 SHA256.

```
#define QAPI_NET_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 0xCC14
```

Cipher TLS ECDHE ECDSA with CHACHA20 POLY1305 SHA256.

```
#define QAPI_NET_TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC15
```

A2560xCC15

Cipher TLS DHE RSA with CHACHA20 POLY1305 SHA256.

```
#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_Vailidity	TRUE to verify certificate time validity.
uint8_t	send_Alert	TRUE to immediately send a fatal alert on detection of an untrusted certificate.
char	match_Name	Name to match against the common name or altDNSNames of the certificate. See QAPI_NET_SSL_MAX_CERT_NAME_LEN .

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 QAPI_NET_SSL_PROTOCOL_*.
uint16_t	cipher	Cipher to use. See SSL cipher suites QAPI_NET_TLS* and QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH .
qapi_Net_SSL_Verify_Policy_t	verify	Certificate verification policy.
uint16_t	max_Frag_Len	Maximum fragment length in bytes.
uint16_t	max_Frag_Len_Neg_Disable	Whether maximum fragment length negotiation is allowed. See RFC 6066.
uint16_t	sni_Name_Size	Length of the SNI server name.
char *	sni_Name	Server name for SNI.

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 QAPI_NET_SSL_MAX_NUM_CERTS and QAPI_NET_SSL_MAX_CERT_NAME_LEN .

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

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

Returns

QAPI_OK on success.
QAPI_ERR_INVALID_PARAM_SSL if an error occurred (configuration is invalid).

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 cert_info 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 qapi_Net_IPv4_Route_t arrays in the routing table.
qapi_Net_IPv4_Route_t	route	Array of qapi_Net_IPv4_Route_t 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 qapi_Net_IPv6_Route_t arrays in the routing table.
qapi_Net_IPv6_Route_t	route	Array of type qapi_Net_IPv6_Route_t .

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 qapi_Net_Get_Number_of_Interfaces() if the value is ≥ 1 .
char	interface_Name	Interface name (NULL terminated).
qbool_t	if_Is_Up	TRUE if the interface is up, FALSE if interface is not up (e.g., down or testing).

6.1.2.8 struct qapi_Ping_Info_Resp_s

Pingresponsestructure.

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.

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).

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

Numberofnetworkinterfaces

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

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

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"> QAPI_NET_IPv4CFG_QUERY_E (0) – Get the IPv4 parameters of an interface. QAPI_NET_IPv4CFG_STATIC_IP_E (1) – Assign the IPv4 address, subnet mask, and default gateway.
in	<i>ipv4_Addr</i>	Pointer to the IPv4 address in host order.
in	<i>subnet_Mask</i>	Pointer to the IPv4 subnet mask in host order.
in	<i>gateway</i>	Pointer to the IPv4 gateway address in host order.

Returns

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

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"> • QAPI_NET_ROUTE_ADD_E (0) – Add route. • QAPI_NET_ROUTE_DEL_E (1) – Delete route. • QAPI_NET_ROUTE_SHOW_E (2) – Show route.
in	<i>ipv4_Addr</i>	Pointer to the IPv4 address in host order.
in	<i>subnet_Mask</i>	Pointer to the IPv4 subnet mask in host order.
in	<i>gateway</i>	Pointer to the IPv4 gateway address in host order.
in	<i>route_List</i>	Pointer to the buffer to contain the list of routes, returned with the QAPI_NET_ROUTE_SHOW_E command.

Returns

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

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"> • QAPI_NET_ROUTE_ADD_E (0) – Add route • QAPI_NET_ROUTE_DEL_E (1) – Delete route • QAPI_NET_ROUTE_SHOW_E (2) – Show route
in	<i>ipv6_Addr</i>	Pointer to the IPv6 address.
in	<i>prefix_Length</i>	Pointer to the IPv6 prefix length.
in	<i>next_Hop</i>	Pointer to the IPv6 gateway address.
in	<i>route_List</i>	Pointer to the buffer containing a list of routes, returned with the QAPI_NET_ROUTE_SHOW_E command.

Returns

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

6.15 Get the Interface Scope ID

6.15.1 Function Documentation

qapi_Status_t qapi_Net_IPv6_Get_Scope_ID(const char* *interface_Name*, int32_t * *cope_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

#define MAX_DNS_SVR_NUM4

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


```
#define QAPI_DNS_PORT 53
```

DNS server port.

```
#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.

```
#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 ip46addr	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

```
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.

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.

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

in	<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 ip46addr for the resolved IP address. The caller must specify which IP address (v4 or v6) it intends to resolve to: If ipaddr type is AF_INET, resolve to an IPv4 address. If ipaddr type is AF_INET6, resolve to an IPv6 address.

Returns

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

7.5 Convert an IP Address Text String for an Interface

7.5.1 Function Documentation

· 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 ip46addr for the resolved IP address. The caller must specify which IP address (v4 or v6) it intends to resolve to: If addr type is AF_INET, resolve to an IPv4 address. If addr type is AF_INET6, resolve to an IPv6 address.
in	<i>iface_index</i>	Name of the PDN/APN for which the address text string is to be resolved.

Returns

On success, 0 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

7.11.1.1 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 AF_INET or AF_INET6.

Returns

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

8 HTTP(S) API

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

8.1 HTTP(S) API

8.1.1 Data Structure Documentation

8.1.1.1 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.

8.1.2 Typedef Documentation

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

8.1.3 Enumeration Type Documentation

8.1.3.1 enum `qapi_Net_HTTPc_Method_e`

HTTP request types supported by `qapi_Net_HTTPc_Request()`.

Enumerator:

<i>QAPI_NET_HTTP_CLIENT_GET_E</i>	HTTP	get	request.
<i>QAPI_NET_HTTP_CLIENT_POST_E</i>	HTTP	post	request.
<i>QAPI_NET_HTTP_CLIENT_PUT_E</i>	HTTP	put	request.
<i>QAPI_NET_HTTP_CLIENT_PATCH_E</i>	HTTP	patch	request.

QAPI_NET_HTTP_CLIENT_HEAD_E

HTTP head request.

8.1.3.2 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.

8.1.4 Function Documentation

8.1.4.1 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.

8.1.4.2 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.

8.1.4.3 `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.

8.1.4.4 `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.

8.1.4.5 `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.

8.1.4.6 **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.

8.1.4.7 **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.

8.1.4.8 **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.

8.1.4.9 `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 databuffer.

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.

8.1.4.10 `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.

8.1.4.11 `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.

8.1.4.12 `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.

9 QAPI Status and Error Codes

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

9.1 QAPI Status Codes

9.1.1 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)`

9.1.2 Generic Error Codes

- #define `QAPI_NET_ERR_INVALID_IPADDR`
`((qapi_Status_t)(QAPI_ERROR(QAPI_MOD_N- ETWORKING,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)))`

9.1.3 MQTT Error Codes

- #define `QAPI_NET_MQTT_ERR_NUM_START` 25

- #define [QAPI_NET_MQTT_ERR_ALLOC_FAILURE](#)((qapi_Status_t)
QAPI_ERROR(QAPI_M- OD_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_MO- D_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+3))
- #define [QAPI_NET_MQTT_ERR_MSG_DESERIALIZATION_FAILURE](#)((qapi_Status_t)
QAP- I_ERROR(QAPI_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+4))
- #define [QAPI_NET_MQTT_ERR_MSG_SERIALIZATION_FAILURE](#)((qapi_Status_t)
QAPI_E- RROR(QAPI_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#) +
5))
- #define [QAPI_NET_MQTT_ERR_NEGATIVE_CONNACK](#)((qapi_Status_t)
QAPI_ERROR(Q- API_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+6))
- #define [QAPI_NET_MQTT_ERR_NO_DATA](#)((qapi_Status_t)
QAPI_ERROR(QAPI_MOD_NE- TWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+7))
- #define [QAPI_NET_MQTT_ERR_NONZERO_REFCOUNT](#) ((qapi_Status_t)
QAPI_ERROR(Q- API_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+8))
- #define [QAPI_NET_MQTT_ERR_PINGREQ_MSG_CREATION_FAILED](#)((qapi_Status_t)
QA- PI_ERROR(QAPI_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+9))
- #define [QAPI_NET_MQTT_ERR_PUBACK_MSG_CREATION_FAILED](#)((qapi_Status_t)
QAP- I_ERROR(QAPI_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+10))
- #define [QAPI_NET_MQTT_ERR_PUBCOMP_MSG_CREATION_FAILED](#)((qapi_Status_t)
Q- API_ERROR(QAPI_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+11))
- #define [QAPI_NET_MQTT_ERR_PUBLISH_MSG_CREATION_FAILED](#)((qapi_Status_t)
QA- PI_ERROR(QAPI_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+12))
- #define [QAPI_NET_MQTT_ERR_PUBREC_MSG_CREATION_FAILED](#)((qapi_Status_t)
QAP- I_ERROR(QAPI_MOD_NETWORKING, [QAPI_NET_MQTT_ERR_NUM_START](#)
+13))

- #define `QAPI_NET_MQTT_ERR_PUBREL_MSG_CREATION_FAILED`((qapi_Status_t)
QAPI_ I_ERROR(QAPI_MOD_NETWORKING, `QAPI_NET_MQTT_ERR_NUM_START`
+14))
- #define `QAPI_NET_MQTT_ERR_RX_INCOMPLETE`((qapi_Status_t)
QAPI_ERROR(QAPI_ - MOD_NETWORKING, `QAPI_NET_MQTT_ERR_NUM_START`
+15))
- #define `QAPI_NET_MQTT_ERR_SOCKET_FATAL_ERROR` ((qapi_Status_t)
QAPI_ERROR(- QAPI_MOD_NETWORKING, `QAPI_NET_MQTT_ERR_NUM_START` +
16))
- #define `QAPI_NET_MQTT_ERR_TCP_BIND_FAILED`((qapi_Status_t)
QAPI_ERROR(QAPI_ - MOD_NETWORKING, `QAPI_NET_MQTT_ERR_NUM_START`
+17))
- #define `QAPI_NET_MQTT_ERR_TCP_CONNECT_FAILED`((qapi_Status_t)
QAPI_ERROR(Q- API_MOD_NETWORKING, `QAPI_NET_MQTT_ERR_NUM_START`
+18))
- #define `QAPI_NET_MQTT_ERR_SSL_CONN_FAILURE`((qapi_Status_t)
QAPI_ERROR(QAP- I_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_E- RROR(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)
QA- PI_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

9.1.4 QAPI Modules

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

Note that if OEMs wish to added 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_CRYPT0 (14)`

9.1.5 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)))

9.2 Define Documentation

#define QAPI_ERR_SSL_CERT QAPI_ERROR(QAPI_MOD_NETWORKING,1)

Error in own certificate.

#define QAPI_ERR_SSL_CONN QAPI_ERROR(QAPI_MOD_NETWORKING,2)

Error with the SSL connection.

#define QAPI_ERR_SSL_HS_NOT_DONE QAPI_ERROR(QAPI_MOD_NETWORKING,3)

Handshake must be completed before the operation can be attempted.

#define QAPI_ERR_SSL_ALERT_RECV QAPI_ERROR(QAPI_MOD_NETWORKING,4)

Received an SSL warning alert message.

**#define QAPI_ERR_SSL_ALERT_FATAL
QAPI_ERROR(QAPI_MOD_NETWORKING,5)**

Received an SSL fatal alert message.

#define QAPI_SSL_HS_IN_PROGRESS QAPI_ERROR(QAPI_MOD_NETWORKING,6)

Handshake is in progress.

#define QAPI_SSL_OK_HS QAPI_ERROR(QAPI_MOD_NETWORKING,7)

Handshake was successful.

#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.

#define QAPI_ERR_SSL_CERT_TIME

QAPI_ERROR(QAPI_MOD_NETWORK- KING, 9)

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

```
#define QAPI_ERR_SSL_CERT_NONE    QAPI_ERROR(QAPI_MOD_NETWORK-  
ORNING, 10)
```

The SSL certificate of the peer is not trusted.

```
#define QAPI_ERR_SSL_NETBUF    QAPI_ERROR(QAPI_MOD_NETWORKKI-  
NG, 11)
```

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

```
#define QAPI_ERR_SSL SOCK    QAPI_ERROR(QAPI_MOD_NETWORKING,  
12)
```

Socket error.

```
#define QAPI_NET_ERR_INVALID_IPADDR ((qapi_Status_t)(    QAPI_ERRO-  
R(QAPI_MOD_NETWORKING,    21)))
```

IP address is invalid.

```
#define QAPI_NET_ERR_CANNOT_GET_SCOPEID ((qapi_Status_t)(    QAPI-  
_ERROR(QAPI_MOD_NETWORKING,    22)))
```

Failed to get the scope ID.

```
#define QAPI_NET_ERR_SOCKET_CMD_TIME_OUT ((qapi_Status_t)(    QA-  
PI_ERROR(QAPI_MOD_NETWORKING,    23)))
```

Socket command timed out.

```
#define QAPI_NET_ERR_MAX_SERVER_REACHED ((qapi_Status_t)(    QAP-  
I_ERROR(QAPI_MOD_NETWORKING,    24)))
```

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

```
#define    QAPI_NET_MQTT_ERR_NUM_START    25
```

MQTT error number start.

```
#define QAPI_NET_MQTT_ERR_ALLOC_FAILURE ((qapi_Status_t)    QAPI_-  
ERROR(QAPI_MOD_NETWORKING,  
QAPI_NET_MQTT_ERR_NUM_START)
```

Memory allocation failed.

```
#define QAPI_NET_MQTT_ERR_BAD_PARAM ((qapi_Status_t)    QAPI_ER-  
ROR(QAPI_MOD_NETWORKING,  
QAPI_NET_MQTT_ERR_NUM_START    + 1))
```

Bad parameter while invoking the API.

```
#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.

```
#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.

```
#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.

```
#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.

```
#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.

```
#define QAPI_NET_MQTT_ERR_NO_DATA ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,
QAPI_NET_MQTT_ERR_NUM_START + 7))
```

No data.

```
#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.

```
#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.

```
#define QAPI_NET_MQTT_ERR_PUBACK_MSG_CREATION_FAILED
```

```
LED ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,  
QAPI_NET_MQTT_ERR_NUM_START + 10))
```

PUBACK message creation failed.

```
#define QAPI_NET_MQTT_ERR_PUBCOMP_MSG_CREATION_FAI-  
ILED ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,  
QAPI_NET_MQTT_ERR_NUM_START + 11))
```

PUBCOMP message creation failed.

```
#define QAPI_NET_MQTT_ERR_PUBLISH_MSG_CREATION_FAI-  
LED ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,  
QAPI_NET_MQTT_ERR_NUM_START + 12))
```

Publish message creation failed.

```
#define QAPI_NET_MQTT_ERR_PUBREC_MSG_CREATION_FAI-  
LED ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,  
QAPI_NET_MQTT_ERR_NUM_START + 13))
```

PUBREC message creation failed.

```
#define QAPI_NET_MQTT_ERR_PUBREL_MSG_CREATION_FAI-  
LED ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,  
QAPI_NET_MQTT_ERR_NUM_START + 14))
```

PUBREL message creation failed.

```
#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.

```
#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.

```
#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.

```
#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.

```
#define QAPI_NET_MQTT_ERR_SSL_CONN_FAILURE ((qapi_Status_t) Q-
API_ERROR(QAPI_MOD_NETWORKING,
QAPI_NET_MQTT_ERR_NUM_ST- ART + 19))
```

SSL connection failed.

```
#define QAPI_NET_MQTT_ERR_SUBSCRIBE_MSG_CREATION_FA-
ILED ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,
QAPI_NET_MQTT_ERR_NUM_START + 21))
```

Subscribe message creation failed.

```
#define QAPI_NET_MQTT_ERR_SUBSCRIBE_UNKNOWN_TOPIC ((qapi-
Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,
QAPI_NET_MQTT_E- RR_NUM_START + 21))
```

Subscribe unknown topic.

```
#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.

```
#define QAPI_NET_MQTT_ERR_UNEXPECTED_MSG ((qapi_Status_t) QA-
PI_ERROR(QAPI_MOD_NETWORKING,
QAPI_NET_MQTT_ERR_NUM_STA- RT + 23))
```

Unexpected message was received.

```
#define QAPI_NET_MQTT_ERR_PARTIAL_SUBSCRIPTION_FAIL-
URE ((qapi_Status_t) QAPI_ERROR(QAPI_MOD_NETWORKING,
QAPI_NET_MQTT_ERR_NUM_START + 24))
```

Subscription failed.

```
#define QAPI_NET_MQTT_ERR_RESTORE_FAILURE ((qapi_Status_t) QA-
PI_ERROR(QAPI_MOD_NETWORKING,
QAPI_NET_MQTT_ERR_NUM_STA- RT + 25))
```

Restore failed.

```
#define QAPI_NET_MQTT_ERR_MAX_NUMS 26
```

Maximum error number.

```
#define QAPI_OK((qapi_Status_t)(0))
```

Success.

```
#define QAPI_ERROR((qapi_Status_t)( QAPI_ERROR(QAPI_MOD_BASE,  
1)))
```

General error.

```
#define QAPI_ERR_INVALID_PARAM((qapi_Status_t)( QAPI_ERROR(QA-  
PI_MOD_BASE,2)))
```

Invalid parameter.

```
#define QAPI_ERR_NO_MEMORY((qapi_Status_t)( QAPI_ERROR(QAPI_-  
MOD_BASE,3)))
```

Memory allocation error.

```
#define QAPI_ERR_NO_RESOURCE((qapi_Status_t)( QAPI_ERROR(QAPI-  
_MOD_BASE,4)))
```

Resource allocation error.

```
#define QAPI_ERR_BUSY((qapi_Status_t)( QAPI_ERROR(QAPI_MOD_BA-  
SE,6)))
```

Operation is busy.

```
#define QAPI_ERR_NO_ENTRY((qapi_Status_t)( QAPI_ERROR(QAPI_MO-  
D_BASE,7)))
```

Entry was not found.

```
#define QAPI_ERR_NOT_SUPPORTED((qapi_Status_t)( QAPI_ERROR(Q-  
API_MOD_BASE, 8)))
```

Feature is not supported.

```
#define QAPI_ERR_TIMEOUT((qapi_Status_t)( QAPI_ERROR(QAPI_MOD-  
_BASE,9)))
```

Operation timed out.

```
#define QAPI_ERR_BOUNDS ((qapi_Status_t)( QAPI_ERROR(QAPI_MOD_-  
BASE, 10)))
```

Out of bounds.

```
#define QAPI_ERR_BAD_PAYLOAD ((qapi_Status_t)( QAPI_ERROR(QAPI-  
_MOD_BASE, 11)))
```

Bad payload.

```
#define QAPI_ERR_EXISTS ((qapi_Status_t)( QAPI_ERROR(QAPI_MOD_B-  
ASE, 12)))
```

Entry already exists.

10 System Drivers APIs

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

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

10.1 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 function
                                          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!
}
```

support gpio interrupt pin name:

PCM_CLK
PCM_SYNC
PCM_DIN
PCM_DOUT
I2C_SCL
GPIO4
STATUS
GPIO3

10.1.1 Typedef Documentation

10.1.1.1 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.

10.1.1.2 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.

10.1.1.3 typedef void* qapi_Instance_Handle_t

GPIO interrupt handle definition.

10.1.2 Enumeration Type Documentation

10.1.2.1 enum qapi_GPIOINT_Trigger_e

GPIO interrupt trigger type enumeration for supported triggers.

Enumerator:

<i>QAPI_GPIOINT_TRIGGER_LEVEL_HIGH_E</i>	Level triggered active high.
<i>QAPI_GPIOINT_TRIGGER_LEVEL_LOW_E</i>	Level triggered active low.
<i>QAPI_GPIOINT_TRIGGER_EDGE_RISING_E</i>	Rising edge triggered.
<i>QAPI_GPIOINT_TRIGGER_EDGE_FALLING_E</i>	Falling edge triggered.
<i>QAPI_GPIOINT_TRIGGER_EDGE_DUAL_E</i>	Dual edge triggered.

10.1.2.2 enumqapi_GPIOINT_Priority_e

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

Enumerator:

<i>QAPI_GPIOINT_PRIO_HIGHEST_E</i>	Highest priority.
<i>QAPI_GPIOINT_PRIO_HIGH_E</i>	Medium-high priority.
<i>QAPI_GPIOINT_PRIO_MEDIUM_E</i>	Medium priority.
<i>QAPI_GPIOINT_PRIO_LOW_E</i>	Medium-low priority.
<i>QAPI_GPIOINT_PRIO_LOWEST_E</i>	Highestpriority.

10.1.3 Function Documentation

10.1.3.1 qapi_Status_t qapi_GPIOINT_Register_Interrupt (qapi_Instance_Handle_t pH, uint32_t nGpio, qapi_GPIOINT_CB_t pfnCallback, qapi_G- PPOINT_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.

Please confirm that the direction of the GPIO pin is INPUT.

If the GPIO pin is high, Trigger type do not set as *QAPI_GPIOINT_TRIGGER_LEVEL_HIGH_E*.

If the GPIO pin is low, Trigger type do not set as *QAPI_GPIOINT_TRIGGER_LEVEL_LOW_E*.

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.
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.

10.1.3.2 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.

10.1.3.3 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.

10.1.3.4 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	pH	Input handle to the client context.
in	nGpio	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.

10.1.3.5 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	pH	Input handle to the client context.
in	nGpio	GPIO number to disable.

Returns

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

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

10.1.3.6 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.

10.1.3.7 `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.

10.2 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;//pin number
tlmm_config.func      = 0;
tlmm_config.dir       =
QAPI_GPIO_OUTPUT_E;
tlmm_config.pull      = API_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
    }
}
```

You can use these gpio pins:

PCM_CLK
PCM_SYNC
PCM_DIN
PCM_DOUT
I2C_SDA
I2C_SCL
GPIO4
NETLIGHT
STATUS
GPIO2
GPIO3

10.2.1 Data Structure Documentation

10.2.1.1 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.

10.2.2 Typedef Documentation

10.2.2.1 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.

10.2.3 Enumeration Type Documentation

10.2.3.1 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.

10.2.3.2 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.

10.2.3.3 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.

<i>QAPI_GPIO_4MA_E</i>	Specify a 4 mA drive.
<i>QAPI_GPIO_6MA_E</i>	Specify a 6 mA drive.
<i>QAPI_GPIO_8MA_E</i>	Specify an 8 mA drive.
<i>QAPI_GPIO_10MA_E</i>	Specify a 10 mA drive.
<i>QAPI_GPIO_12MA_E</i>	Specify a 12 mA drive.
<i>QAPI_GPIO_14MA_E</i>	Specify a 14 mA drive.
<i>QAPI_GPIO_16MA_E</i>	Specify a 16 mA drive.

10.2.3.4 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:

<i>QAPI_GPIO_LOW_VALUE_E</i>	Drive the output LOW.
<i>QAPI_GPIO_HIGH_VALUE_E</i>	Drive the output HIGH.

10.2.4 Function Documentation

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

Returns

- QAPI_OK – Pin GPIO ID was successfully created.
- QAPI_ERR – Pin GPIO is currently in use or not programmable.

10.2.4.2 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 qapi_TLMM_Get_Gpio_ID() call.
-----------	---------------------	--

Returns

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

10.2.4.3 `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 qapi_TLMM_Get_Gpio_ID() call.
in	<i>qapi_TLMM_Config</i>	Pin configuration to use.

Returns

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

10.2.4.4 `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 qapi_TLMM_Get_Gpio_ID() call.
in	<i>pin</i>	SoC pin number to configure.
in	<i>value</i>	Output value.

Returns

- QAPI_OK – Operation completed successfully.
- QAPI_ERR – Operation failed.

10.2.4.5 `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 qapi_TLMM_Get_Gpio_ID() call.
in	<i>pin</i>	SoC pin number to configure.
out	<i>qapi_GPIO_Value</i>	GPIO pin value.

Returns

- QAPI_GPIO_HIGH_VALUE – Read value was HIGH.
- QAPI_GPIO_LOW_VALUE – Read value was LOW.

11 Storage Module

This chapter describes the file system data types and APIs.

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

11.1 File System Data Types

11.1.1 Data Structure Documentation

11.1.1.1 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.

11.1.1.2 structqapi_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 qapi_F-S_Filename_-Rule_e	f_name_rule	File naming rule.
enum qapi_F-S_Filename_-Encoding_e	f_name_-encoding	Encoding scheme.

11.1.1.3 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 qapi_FS-Stat_Type_s	SBuf	See qapi_FS_Stat_Type_s for information on this structure.
uint32	qapi_FS_D_-Stats_Present	Bitmask for the qapi_FS_Stat_Type_s structure that defines which fields are filled when the qapi_FS_Iter_Next() API is called.

11.1.2 Enumeration Type Documentation

11.1.2.1 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.

11.1.2.2 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.

11.2 File System APIs

11.2.1 Function Documentation

11.2.1.1 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	Path	Path of the file that is to be opened.
----	------	--

in	<i>Oflag</i>	<p>Argument that describes how this file is to be opened. It contains one of the following values:</p> <ul style="list-style-type: none"> • QAPI_FS_O_RDONLY_E – Open for read only. • QAPI_FS_O_WRONLY_E – Open for write only. • QAPI_FS_O_RDWR_E – Open for read and write. In addition, the following flags can be bitwise ORed with this value: • QAPI_FS_O_APPEND_E – All writes will self-seeking to the end of the file before writing. • QAPI_FS_O_CREAT_E – Create the file if it does not exist. • QAPI_FS_O_TRUNC_E – Truncate the file to zero bytes on successful open. The following flags can be used to specify common ways of opening files: • QAPI_FS_O_CREAT_E QAPI_FS_O_TRUNC_E – Normal for writing a file. Creates it if it does not exist. The resulting file is zero bytes long. • QAPI_FS_O_CREAT_E QAPI_FS_O_EXCL_E – Creates a file but fails if it already exists.
in	<i>Mode</i>	<p>If QAPI_FS_O_CREAT is a part of Oflag, a third argument (Mode) must be passed to qapi_FS_Open() to define the file permissions given to the newly created file. If QAPI_FS_O_CREAT is not a part of flag, set Mode=0.</p> <p>One or more of the following permission bits can be ORed as the Mode parameter:</p> <ul style="list-style-type: none"> • QAPI_FS_S_IRUSR_E – Read permission for a user • QAPI_FS_S_IWUSR_E – Write permission for a user • QAPI_FS_S_IXUSR_E – Execute permission for a user • QAPI_FS_S_IRGRP_E – Read permission for a group • QAPI_FS_S_IWGRP_E – Write permission for a group • QAPI_FS_S_IXGRP_E – Execute permission for a group • QAPI_FS_S_IROTH_E – Read permission for other • QAPI_FS_S_IWOTH_E – Write permission for other • QAPI_FS_S_IXOTH_E – Execute permission for other • QAPI_FS_S_ISUID_E – Set UID on execution • QAPI_FS_S_ISGID_E – Set GID on execution • QAPI_FS_S_ISVTX_E – Sticky bit (hidden attribute in FAT)
out	<i>Fd_ptr</i>	<p>Address of the file descriptor variable. On success, the file descriptor of an opened file is written to it. On failure, the variable is set to -1.</p>

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

- QAPI_ERR_EXISTS – 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_NO_ENTRY – 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_NO_MEMORY – 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_INVALID_PARAM – Invalid parameter or path.

11.2.1.2 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 qapi_FS_Open_With_Mode() .
out	<i>Fd_ptr</i>	Address of the file descriptor variable. On success, the file descriptor of an opened file is written to it. On failure, the variable is set to -1.

Returns

See [qapi_FS_Open_With_Mode\(\)](#).

11.2.1.3 `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 qapi_FS_Open() function.
out	<i>Buf</i>	Buffer where the read bytes from the file will be stored.
in	<i>Count</i>	Number of bytes to read from the file.
out	<i>Bytes_Read_Ptr</i>	This is a return from the function with the number of bytes actually read.

Returns

Returns QAPI_OK on success, and -ve error code is returned on failure.

11.2.1.4 `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 ioperation may happen at the current file pointer or at the end of the file if the file is opened with QAPI_FS_O_APPEND. It is permitted for `qapi_FS_Write` to write fewer bytes than were requested, even if space is available. If the number of bytes written is zero, it indicates that the file system is full (writing), which will result in an QAPI_ERR_ENOSPC error.

Parameters

in	<i>Fd</i>	File descriptor obtained via the qapi_FS_Open() function.
in	<i>Buf</i>	Buffer to which the file is to be written.
in	<i>Count</i>	Number of bytes to write to the file.
out	<i>Bytes_Written_Ptr</i>	This is a return from the function with the number of bytes actually written.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

11.2.1.5 `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 <code>qapi_FS_Open()</code> function.
-----------	-----------	--

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

11.2.1.6 `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 across different file systems is not permitted.
- QAPI_ERR_ENOTEMPTY – The *Old_Path* directory is not empty.

11.2.1.7 `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.

11.2.1.8 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 qapi_FS_Open() API.
in	<i>Offset</i>	New offset of the file.
in	<i>Whence</i>	Indicates how the new offset is computed: QAPI_FS_SEEK_SET_E – Set to Offset. QAPI_FS_SEEK_CUR_E – Set to Offset + current position. QAPI_FS_SEEK_END_E – Set to Offset + file size.
out	<i>Actual_Offset_Ptr</i>	Upon success, the resulting offset as bytes from the beginning of the file is filled in this parameter. On failure, the variable is set to -1.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

- QAPI_ERR_EINVAL – Invalid operation.

- QAPI_ERR_EBADF – File descriptor was not found.
- QAPI_ERR_ESPIPE – Some file descriptors (like pipes and FIFOs) are not seekable.

11.2.1.9 `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 qapi_FS_Open() API for information on Mode bits.

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

- QAPI_ERR_ENOENT – No 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.

11.2.1.10 `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_NOTEMPTY – The directory is not empty.
- QAPI_ERR_ETXTBSY – The directory is in use or open.
- QAPI_ERR_EINVAL – Invalid parameter.

11.2.1.11 `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 filesystem 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.

11.2.1.12 `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 qapi_FS_Stat_Type_s .

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

11.2.1.13 `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 qapi_FS_Open() API.
out	<i>SBuf</i>	Information is returned in the structure qapi_FS_Stat_Type_s .

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

11.2.1.14 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 qapi_FS_Statvfs_Type_s .

Returns

Returns QAPI_OK on success, and -ve error code is returned on failure.

11.2.1.15 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.

11.2.1.16 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 qapi_FS_Iter_Open() API.
out	<i>Iter_Entry</i>	Details about the next entry found is filled in <pre>struct qapi_FS_Dirent { char file_Path[QAPI_FS_NAME_MAX+1] struct qapi_FS_Stat_Type_s SBuf uint32 qapi_FS_D_Stats_Present; }</pre>

- `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.

11.2.1.17 `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 qapi_FS_Iter_Open() API.
-----------	-----------------	---

Returns

Returns QAPI_OK on success and -ve error code is returned on failure.

11.2.1.18 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.

11.3 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.

11.3.1 Data Structure Documentation

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	ManufacturerID.
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.

11.3.2 Typedef Documentation

11.3.2.1 typedef struct qapi_FTL_info qapi_FTL_info_t

Structure for storing information about a partition.

11.3.2.2 typedef void* qapi_FTL_client_t

Handle returned to the client. One handle is returned per partition.

11.3.3 Function Documentation

11.3.3.1 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 expose any data in and of itself. The handle is to be used with FTL APIs to perform other tasks, e.g., use this handle with [qapi_FTL_Get_info\(\)](#) to get FTL information in the format of [qapi_FTL_info_t](#). As with read and write data functions, this handle must be passed with the correct offset and size.

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.

11.3.3.2 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.

11.3.3.3 `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.

11.3.3.4 `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 <code>qapi_FTL_Open()</code> .
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.

11.3.3.5 `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 <code>qapi_FTL_Open()</code> .
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.

11.3.3.6 `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 qapi_FTL_Open() .
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.

12 Wired Connectivity Module

This chapter describes the USB data types and APIs.

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

12.1 USB Data Types

Type definitions for USB QAPIs.

12.1.1 Data Structure Documentation

12.1.1.1 union `qapi_USB_Ioctl_Param`

IOCTL parameter type.

Data fields

Type	Parameter	Description
qapi_USB_-App_Rx_Cb_t	<code>qapi_USB_-App_Rx_Cb_-Func</code>	Client callback function.

12.1.2 Typedef Documentation

12.1.2.1 typedef `void(* qapi_USB_App_Rx_Cb_t)(void)`

Client callback function type to be called when data is received from the client.

12.1.2.2 typedef union `qapi_USB_Ioctl_Param` `qapi_USB_Ioctl_Param_t`

IOCTL parameter type.

12.1.3 Enumeration Type Documentation

12.1.3.1 enum `qapi_USB_Ioctl_Cmd_t`

IOCTL command type.

Enumerator:

QAPI_USB_RX_CB_REG_E IOCTL command argument to register a client callback.

12.2 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) { ... }
```

12.2.1 Function Documentation

12.2.1.1 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.

12.2.1.2 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.

12.2.1.3 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.

12.2.1.4 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.

13 Buses Module

This chapter describes the I2C, SPI, and UART APIs.

- [I2C Master APIs](#)
- [SPI Master APIs](#)
- [UART APIs](#)

13.1 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 3 res = qapi_I2CM_Open (QAPI_I2CM_INSTANCE_003_E,
    &client_handle);

    // Configure the bus speed and slave address config.
    bus_Frequency_KHz = 400;
    config.slave_Address = 0x1A;
    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 = (uint32_t)&transferred1;
```



```

desc[0].flags = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE |
QAPI_I2C_FLAG_STOP;
result = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 1,
client_callback, (void *)0x11, 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 = (uint32)&transferred1;
desc[0].flags = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ |
QAPI_I2C_FLAG_STOP;
result = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 1,
client_callback, (void *)0x12, 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>
desc[0].buffer = &reg[0];
desc[0].length = 1;
desc[0].transferred = (uint32)&transferred1;
desc[0].flags = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE;
desc[1].buffer = buffer;
desc[1].length = 4;
desc[1].transferred = (uint32)&transferred2;
desc[1].flags = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ |
QAPI_I2C_FLAG_STOP;
result = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 2,
client_callback, (void *)0x13, 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>
desc[0].buffer = reg;
desc[0].length = 2;
desc[0].transferred = (uint32)&transferred1;
desc[0].flags = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE;
desc[1].buffer = buffer;
desc[1].length = 4;
desc[1].transferred = (uint32)&transferred2;
desc[1].flags = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ |
QAPI_I2C_FLAG_STOP;
result = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 2,
client_callback, (void *)0x14, 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
}

```

13.1.1 Define Documentation

```
#define QAPI_I2C_FLAG_START 0x00000001
```

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

```
#define QAPI_I2C_FLAG_STOP 0x00000002
```

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

```
#define QAPI_I2C_FLAG_WRITE 0x00000004
```

Must be set to indicate a WRITE transfer.

```
#define QAPI_I2C_FLAG_READ 0x00000008
```

Must be set to indicate a READ transfer.

```
#define QAPI_I2C_TRANSFER_MASK (QAPI_I2C_FLAG_WRITE | QAPI_I2C_-  
FLAG_READ)
```

Transfer types.

```
#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.

13.1.2 Data Structure Documentation

13.1.2.1 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.
----------	-------------------------	--

13.1.2.2 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.

13.1.3 Typedef Documentation

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

13.1.4 Enumeration Type Documentation

13.1.4.1 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.

13.1.5 Function Documentation

13.1.5.1 `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 qapi_I2CM_Instance_t for details.
out	<i>i2c_Handle</i>	Pointer location to be filled by the driver with a handle to the instance.

Returns

- **QAPI_OK** – Function was successful.
- **QAPI_I2CM_ERR_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.

13.1.5.2 `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.

13.1.5.3 `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 qapi_I2CM_Config_t for details.
in	<i>desc</i>	I2C transfer descriptor. See qapi_I2CM_Descriptor_t for details. This can be an array of descriptors.
in	<i>num_Descriptors</i>	Number of descriptors in the descriptor array.
in	<i>CB_Function</i>	Callback function that is called at the completion of the transfer occurs in the interrupt context. The call must do minimal processing and must not call any API defined here.
in	<i>CB_Parameter</i>	Context that the client passes here is returned as is in the callback function.
in	<i>delay_us</i>	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.

13.1.5.4 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 qapi_I2CM_Open() .
-----------	-------------------	--

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.

13.1.5.5 `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 qapi_I2CM_Open() .
-----------	-------------------	--

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.

13.2 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

13.2.1 Data Structure Documentation

13.2.1.1 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
qapi_SPIM_Shift_Mode_t	SPIM_Mode	SPIM mode type to be used for the SPI core.
qapi_SPIM_CS_Polarity_t	SPIM_CS_Polarity	CS polarity type to be used for the SPI core.
qapi_SPIM_Byte_Order_t	SPIM_endianness	
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.
qapi_SPIM_CS_Mode_t	SPIM_CS_Mode	CS mode to be used for the SPI core.
qbool_t	loopback_Mode	Normally 0. If set, the SPI controller will enable Loopback mode; used primarily for testing.

13.2.1.2 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.

13.2.2 Typedef Documentation

13.2.2.1 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.

13.2.3 Enumeration Type Documentation

13.2.3.1 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:

<i>QAPI_SPIM_INSTANCE_1_E</i>	SPIM instance 1.
<i>QAPI_SPIM_INSTANCE_2_E</i>	SPIM instance 2.
<i>QAPI_SPIM_INSTANCE_3_E</i>	SPIM instance 3.
<i>QAPI_SPIM_INSTANCE_4_E</i>	SPIM instance 4.
<i>QAPI_SPIM_INSTANCE_5_E</i>	SPIM instance 5.
<i>QAPI_SPIM_INSTANCE_6_E</i>	SPIM instance 6.
<i>QAPI_SPIM_INSTANCE_7_E</i>	SPIM instance 7.
<i>QAPI_SPIM_INSTANCE_8_E</i>	SPIM instance 8.
<i>QAPI_SPIM_INSTANCE_9_E</i>	SPIM instance 9.
<i>QAPI_SPIM_INSTANCE_10_E</i>	SPIM instance 10.
<i>QAPI_SPIM_INSTANCE_11_E</i>	SPIM instance 11.
<i>QAPI_SPIM_INSTANCE_12_E</i>	SPIM instance 12.
<i>QAPI_SPIM_INSTANCE_13_E</i>	SPIM instance 13.
<i>QAPI_SPIM_INSTANCE_14_E</i>	SPIM instance 14.
<i>QAPI_SPIM_INSTANCE_15_E</i>	SPIM instance 15.
<i>QAPI_SPIM_INSTANCE_16_E</i>	SPIM instance 16.
<i>QAPI_SPIM_INSTANCE_17_E</i>	SPIM instance 17.
<i>QAPI_SPIM_INSTANCE_18_E</i>	SPIM instance 18.
<i>QAPI_SPIM_INSTANCE_19_E</i>	SPIM instance 19.
<i>QAPI_SPIM_INSTANCE_20_E</i>	SPIM instance 20.
<i>QAPI_SPIM_INSTANCE_21_E</i>	SPIM instance 21.
<i>QAPI_SPIM_INSTANCE_22_E</i>	SPIM instance 22.
<i>QAPI_SPIM_INSTANCE_23_E</i>	SPIM instance 23.
<i>QAPI_SPIM_INSTANCE_24_E</i>	SPIM instance 24.

13.2.3.2 enumqapi_SPIM_Shift_Mode_t

SPI phase type.

This type defines the clock phase that the client can set in the SPI configuration.

Enumerator:

<i>QAPI_SPIM_MODE_0_E</i>	CPOL = 0, CPHA = 0.
<i>QAPI_SPIM_MODE_1_E</i>	CPOL = 0, CPHA = 1.
<i>QAPI_SPIM_MODE_2_E</i>	CPOL = 1, CPHA = 0.
<i>QAPI_SPIM_MODE_3_E</i>	CPOL = 1, CPHA = 1.

13.2.3.3 enumqapi_SPIM_CS_Polarity_t

SPI chip select ppolarity 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.

13.2.3.4 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_ENGIAN Big Endian (network).

13.2.3.5 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.

13.2.4 Function Documentation

13.2.4.1 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 qapi_SPIM_Instance_t .
out	<i>spi_Handle</i>	Pointer to a location in which to store the driver handle.

Returns

QAPI_OK – Module initialized successfully.

QAPI_SPIM_ERROR_INVALID_PARAM – Invalid instance or handle parameter.

QAPI_SPIM_ERROR_MEM_ALLOC – Could not allocate space for driver structures.

QAPI_SPIM_ERR_INTERRUPT_REGISTER – Could not register for an interrupt.

13.2.4.2 `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 qapi_SPIM_Open() .
-----------	-------------------	--

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.

13.2.4.3 `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 qapi_SPIM_Open() .
-----------	-------------------	--

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.

13.2.4.4 `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 qapi_SPIM_Open() .
in	<i>config</i>	Pointer to the SPI configuration structure described by qapi_SPIM_Config_t .
in	<i>desc</i>	Pointer to the structure described by qapi_SPIM_Descriptor_t . 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.

13.2.4.5 `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 qapi_SPIM_Open() .
-----------	-------------------	--

Returns

QAPI_OK – SPI driver closed successfully.

13.3 UART APIs

This section describes the UART data types and APIs.

13.3.1 Data Structure Documentation

13.3.1.1 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.
QAPI_Flow_Control_Type	Flow_Control_Type	Transmit flow control type.

13.3.1.2 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.
qapi_UART_Parity_Mode_e	parity_Mode	Parity mode.
qapi_UART_Num_Stop_Bits_e	num_Stop_Bits	Number of stop bits.
qapi_UART_Bits_Per_Char_e	bits_Per_Char	Bits per character.
qbool_t	enable_Loopback	Enable loopback.
qbool_t	enable_Flow_Ctrl	Enable flow control.
qapi_UART_Callback_Fn_t	tx_CB_ISR	Transmit callback, called from ISR context. Be sure not to violate ISR guidelines. Note: Do not call uart_transmit or uart_receive APIs from this callback.
qapi_UART_Callback_Fn_t	rx_CB_ISR	Receive callback, called from ISR context. Be sure not to violate ISR guidelines. Note: Do not call uart_transmit or uart_receive APIs from this callback.

13.3.2 Typedef Documentation

13.3.2.1 typedef union QAPI_UART_Ioctl_Param QAPI_UART_Ioctl_Param

IOCTL command ID list of the UART.

13.3.2.2 typedef void* qapi_UART_Handle_t

UART handle that is passed to the client when a UART port is opened.

13.3.2.3 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.

13.3.3 Enumeration Type Documentation

13.3.3.1 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.

13.3.3.2 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.

13.3.3.3 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.

13.3.3.4 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.

13.3.3.5 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.

13.3.3.6 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.

13.3.4 Function Documentation

13.3.4.1 `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 qapi_UART_Open() .
-----------	---------------	--

Returns

QAPI_OK Port close successful.

QAPI_ERROR Port close failed.

13.3.4.2 `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.

13.3.4.3 `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 qapi_UART_Open() .
in	<i>buf</i>	Buffer to be filled with data.
in	<i>buf_Size</i>	Buffer size. Must be ≥ 4 and a multiple of 4.
in	<i>cb_Data</i>	Callback data to be passed when <code>rx_cb_isr</code> is called during Rx completion.

Returns

QAPI_OK Queuing of the receive buffer was successful.

QAPI_ERROR Queuing of the receive buffer failed.

13.3.4.4 `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 qapi_UART_Open() .
in	<i>buf</i>	Buffer with data for transmit.
in	<i>bytes_To_Tx</i>	Bytes of data to transmit.
in	<i>cb_Data</i>	Callback data to be passed when <code>tx_cb_isr</code> is called during Tx completion.

Returns

QAPI_OK Queuing of the transmit buffer was successful.

QAPI_ERROR Queuing of the transmit buffer failed.

13.3.4.5 `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	<i>UART_Handle</i>	Driver handle returned by qapi_UART_Open() .
-----------	--------------------	--

Returns

QAPI_OK UART powered on successfully.

QAPI_ERROR UART power on is failed.

13.3.4.6 `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	<i>UART_Handle</i>	Driver handle returned by qapi_UART_Open() .
-----------	--------------------	--

Returns

QAPI_OK UART powered off successfully.

QAPI_ERROR UART power off is failed.

13.3.4.7 `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 qapi_UART_Open() .
in	<i>ioctl_Command</i>	Pass the commands listed with qapi_UART_Ioctl_Command_e .

in	<i>ioctl_Param</i>	Pass the argument associated with qapi_UART_Ioctl_Command_e .
-----------	--------------------	---

Returns

QAPI_OK UART IOCTL configuration is successfull.

QAPI_ERROR UART IOCTL configuration is failed.

14 Location Module

This chapter describes the data types and APIs for the GNSS location driver.

- [Location APIs](#)

14.1 Location APIs

This section describes data types and functions for the GNSS location driver.

14.1.1 Data Structure Documentation

14.1.1.1 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.

14.1.1.2 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	dwelTime	Dwell time in seconds.

14.1.1.3 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.

14.1.1.4 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.

14.1.1.5 struct `qapi_Location_Callbacks_t`

Location callbacks requirements.

Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of qapi_Location_Callbacks_t .
qapi_Capabilities_Callback	capabilitiesCb	Capabilities callback is mandatory.
qapi_Response_Callback	responseCb	Response callback is mandatory.
qapi_Geofence_Response_Callback	geofence-ResponseCb	Geofence response callback is mandatory.
qapi_Tracking_Callback	trackingCb	Tracking callback is optional.
qapi_Batching_Callback	batchingCb	Batching callback is optional.
qapi_Geofence_Breach_Callback	geofence-BreachCb	Geofence breach callback is optional.

14.1.2 Typedef Documentation

14.1.2.1 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 qapi_Location_Capabilities_Mask_t .
-----------	-------------------------	---

Returns

None.

14.1.2.2 `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.

14.1.2.3 `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 qapi_Location_Error_t associated with the request.
in	<i>ids</i>	Array of IDs to be associated with the request.

Returns

None.

14.1.2.4 `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.

14.1.2.5 `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.

14.1.2.6 `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

14.1.3 Enumeration Type Documentation

14.1.3.1 `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.

14.1.3.2 `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.

14.1.3.3 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.

14.1.3.4 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.

14.1.3.5 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.

14.1.4 Function Documentation

14.1.4.1 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.

14.1.4.2 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.

14.1.4.3 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"> • minInterval – Minimum interval between fixes • minDistance – Minimum distance to travel before reporting
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.

14.1.4.4 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.

14.1.4.5 `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"> • minInterval – Minimum interval between fixes • minDistance – Minimum distance to travel before reporting

Returns

- QAPI_LOCATION_ERROR_SUCCESS if successful.
- QAPI_LOCATION_ERROR_NOT_INITIALIZED if no location session has been initialized.

14.1.4.6 `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"> • minInterval – Minimum interval between fixes • minDistance – Minimum distance to travel before reporting
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.

14.1.4.7 `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.

14.1.4.8 `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"> • <code>minInterval</code> – Minimum interval between fixes • <code>minDistance</code> – Minimum distance to travel before reporting

Returns

- `QAPI_LOCATION_ERROR_SUCCESS` if successful.
- `QAPI_LOCATION_ERROR_NOT_INITIALIZED` if no location session has been initialized.

14.1.4.9 `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.

14.1.4.10 `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"> • breachTypeMask – Bitwise OR of GeofenceBreachTypeMask bits • responsiveness in milliseconds • dwellTime in seconds
in	<i>pData</i>	Array of structures containing the data: <ul style="list-style-type: none"> • latitude of the center of the Geofence in degrees • longitude of the center of the Geofence in degrees • radius of the Geofence in meters
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.

14.1.4.11 `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.

14.1.4.12 `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"> • breachTypeMask – Bitwise OR of GeofenceBreachTypeMask bits • responsiveness in milliseconds • dwellTime in seconds

Returns

- QAPI_LOCATION_ERROR_SUCCESS if successful.
- QAPI_LOCATION_ERROR_NOT_INITIALIZED if no location session has been initialized.

14.1.4.13 `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.

14.1.4.14 `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.

15 Timer Module

This chapter describes the timer data types and APIs.

- [Timer APIs](#)
- [PMIC RTC APIs](#)

15.1 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);
```

15.1.1 Data Structure Documentation

15.1.1.1 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.
qapi_TIMER_notify_t	cb_type	Type of notification to receive.
void *	sigs_func_ptr	Specify the signal object or callback function.
uint32_t	sigs_mask_data	Specify the signal mask or callback data.

15.1.1.2 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.
qapi_TIMER_unit_type	unit	Specify units for timer duration.

15.1.1.3 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
qapi_TIMER_info_type	timer_info	Timer information type.
qapi_TIMER_unit_type	unit	Specify units to use for return.

15.1.2 Typedef Documentation

15.1.2.1 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.

15.1.2.2 typedef void(* qapi_TIMER_cb_t)(uint32_t data)

Timer callback type.

Timer callbacks should adhere to this signature.

15.1.3 Enumeration Type Documentation

15.1.3.1 enum qapi_TIMER_notify_t

Timer notification type.

Enumeration of the notifications available on timer expiry.

Enumerator:

<i>QAPI_TIMER_NO_NOTIFY_TYPE</i>	No notification.
<i>QAPI_TIMER_NATIVE_OS_SIGNAL_TYPE</i>	Signal an object.
<i>QAPI_TIMER_FUNC1_CB_TYPE</i>	Call back a function.

15.1.3.2 enum qapi_TIMER_unit_type

Timer unit type.

Enumeration of the units in which timer duration can be specified.

Enumerator:

<i>QAPI_TIMER_UNIT_TICK</i>	Return time in ticks.
<i>QAPI_TIMER_UNIT_USEC</i>	Return time in microseconds.
<i>QAPI_TIMER_UNIT_MSEC</i>	Return time in milliseconds.
<i>QAPI_TIMER_UNIT_SEC</i>	Return time in seconds.
<i>QAPI_TIMER_UNIT_MIN</i>	Return time in minutes.
<i>QAPI_TIMER_UNIT_HOUR</i>	Return time in hours.

15.1.3.3 enum qapi_TIMER_info_type

Timer information type.

Enumeration of the types of information that can be obtained for a timer.

Enumerator:

<i>QAPI_TIMER_TIMER_INFO_ABS_EXPIRY</i>	Return the timetick of timer expiry in native ticks.
<i>QAPI_TIMER_TIMER_INFO_TIMER_DURATION</i>	Return the total duration of the timer in specified units.
<i>QAPI_TIMER_TIMER_INFO_TIMER_REMAINING</i>	Return the remaining duration of the timer in specified units.

15.1.4 Function Documentation

15.1.4.1 `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.

15.1.4.2 `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.

15.1.4.3 `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.

15.1.4.4 **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.

15.1.4.5 **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

15.1.4.6 **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.

15.1.4.7 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.

15.2 PMIC RTC APIs

This module provides the definitions to configure the real-time clock (RTC) alarm peripheral in the power management IC (PMIC).

15.2.1 Data Structure Documentation

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.

15.2.2 Typedef Documentation

```
typedef struct qapi_PM_Rtc_Julian_Type_s qapi_PM_Rtc_Julian_Type_t
```

PMIC's version of the Julian time structure.

15.2.3 Enumeration Type Documentation

15.2.3.1 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.

15.2.3.2 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.

15.2.3.3 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.

15.2.4 Function Documentation

15.2.4.1 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.

15.2.4.2 `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	<i>mode</i>	New RTC time display mode to be used. Valid values (see qapi_PM_Rtc_Display_Type_t): <ul style="list-style-type: none"> • QAPI_PM_RTC_12HR_MODE_E • QAPI_PM_RTC_24HR_MODE_E
-----------	-------------	--

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.

15.2.4.3 `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 [qapi_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 qapi_PM_Rtc_Julian_Type_t 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.

15.2.4.4 **qapi_Status_t** **qapi_PM_Rtc_Alarm_RW_Cmd** (**qapi_PM_Rtc_Cmd_Type_tcmd,**
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"> • QAPI_PM_RTC_SET_CMD_E • QAPI_PM_RTC_GET_CMD_E
in	<i>what_alarm</i>	Alarm type. See qapi_PM_Rtc_Alarm_Type_t .
in	<i>qapi_alarm_time_ptr</i>	Depending on the command, this function will use the structure qapi_PM_Rtc_Julian_Type_t 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.

16 Hardware Engine APIs

This chapter describes the ADC and TSENS data types and APIs.

- [ADC Data Types](#)
- [ADC APIs](#)
- [TSENS Data Types](#)
- [TSENS APIs](#)

16.1 ADC Data Types

16.1.1 Define Documentation

```
#define ADC_INPUT_BATT_ID "BATT_ID"
```

Physical units are in millivolts.

```
#define ADC_INPUT_PA_THERM "PA_THERM"
```

Physical units are in degrees C.

```
#define ADC_INPUT_PA_THERM1 "PA_THERM1"
```

Physical units are in degrees C.

```
#define ADC_INPUT_PMIC_THERM "PMIC_THERM"
```

Physical units are in 0.001 gradients of degrees C.

```
#define ADC_INPUT_VBATT "VBATT"
```

Physical units are in millivolts.

```
#define ADC_INPUT_VPH_PWR "VPH_PWR"
```

Physical units are in millivolts.

```
#define ADC_INPUT_XO_THERM "XO_THERM"
```

Physical units are in 2^{-10} degrees C.

```
#define ADC_INPUT_XO_THERM_GPS "XO_THERM_GPS"
```

Physical units are in 2^{-10} degrees C.

16.1.2 Data Structure Documentation

16.1.2.1 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.

16.1.2.2 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.

16.1.2.3 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.

16.1.2.4 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.

16.1.2.5 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.

16.1.2.6 struct qapi_ADC_Device_Properties_t

ADC device properties structure.

Data fields

Type	Parameter	Description
uint32_t	uNumChannels	Number of ADC channels.

16.1.2.7 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.

16.1.2.8 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.

16.1.2.9 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.

16.1.3 Typedef Documentation

16.1.3.1 `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.

16.1.3.2 `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.

16.1.4 Enumeration Type Documentation

16.1.4.1 `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.

16.2 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.

```

QCLI_Group_Handle_t qcli_adc_handle;
qapi_Status_t status;
qapi_ADC_Handle_t handle;
qapi_Adc_Input_Properties_Type_t Properties;
char *Channel_Name;
uint32_t Channel_Name_Size;
qapi_ADC_Read_Result_t result;
qapi_AdcTM_Input_Properties_Type_t Properties_TM;
qapi_AdcTM_Range_Type_t ADC_TM_Range;
qapi_AdcTM_Request_Params_Type_t ADC_TM_Params;
qapi_ADC_Amp_Threshold_t Threshold_Type;
int32_t TM_Threshold_Set;
int32_t Lower_Tolerance;
int32_t Higher_Tolerance;
int32_t Threshold_Desired;
qapi_AdcTM_Request_Params_Type_t TM_Params_Type_Ptr;

//open ADC
uint32_t id = 0;
status = qapi_ADC_Open(&handle, id);
if (status != QAPI_OK) { ... }

//get ADC Channel Input Properties
//eg: "VBATT"channel:Channel_Name ="VBATT", Channel_Name_Size = strlen
(Channel_Name);
status = qapi_ADC_Get_Input_Properties(handle, Channel_Name, Channel_Name_Size,
&Properties);
if (status != QAPI_OK) { ... }

//read ADC Channel
//eg:"VBATT"channel: Properties.nChannelIdx =0;Properties.nDeviceIdx = 0;
status = qapi_ADC_Read_Channel(handle, &Properties, &result);
//result.microvolts contains the reading

//get ADC TM Channel Input Properties
//eg: "PA_THERM"channel:Channel_Name="PA_THERM",Channel_Name_Size=strlen
(Channel_Name);
status=qapi_ADC_TM_Get_Input_Properties(handle,Channel_Name,Channel_Name_Size,&Prop
erties_TM);
if (status != QAPI_OK) { ... }
else{

    //get TM Channel Range:
    //eg:"PA_THERM"channel:Properties_TM.nChannelIdx=3,
    //Properties_TM.nDeviceIdx =0,
    status=qapi_ADC_Get_Range(handle,&Properties_TM,
                                &ADC_TM_Range);
    if (status != QAPI_OK) { ... }
}

```

```

//set ADC TM Channel Amp Threshold
//eg: "PA_THERM"channel:
//ADC_TM_Params.adcTMInputProps.nChannelIdx =3;
//ADC_TM_Params.adcTMInputProps. nDeviceIdx=0;
//ADC_TM_Params.pfnAdcTMThresholdCb=ADC_TM_Params
_CB;
// Threshold_Type= QAPI_ADC_THRESHOLD_LOWER_E;
// Threshold_Desired=80;
status=qapi_ADC_Set_Amp_Threshold(handle,&ADC_TM_Params,Threshold_Type, &Threshold_Desired,
&TM_Threshold_Set);
if (status != QAPI_OK) { ... }

// set ADC TM Thresholds Enable
//Enable =0/1;
//Threshold_type = QAPI_ADC_THRESHOLD_LOWER_E or
//Threshold_type =QAPI_ADC_THRESHOLD_HIGHER_E;
Status=qapi_ADC_TM_Enable_Thresholds(handle,Enable,
Threshold_type);
if (status != QAPI_OK) { ... }

//set ADC TM Tolerance
//eg: "PA_THERM"channel:
//TM_Params_Type_Ptr.adcTMInputProps.nChannelIdx =3;
//TM_Params_Type_Ptr.adcTMInputProps. nDeviceIdx=0;
//TM_Params_Type_Ptr.pfnAdcTMThresholdCb=ADC_TM_Params_CB;
//Lower_Tolerance = 80;
//Higher_Tolerance=100;
status=qapi_ADC_TM_Set_Tolerance(handle,&TM_Params_Type_Ptr, &Lower_Tolerance, &Higher_Tolerance);
if (status != QAPI_OK) { ... }
}

Void ADC_TM_Params_CB(void *ctxt, const qapi_ADC_Threshold_Result_t *result){
//this is call back method
}

//close ADC
status = qapi_ADC_Close(handle, false);
if (status != QAPI_OK) { ... }
handle = NULL;

```

16.2.1 Function Documentation

16.2.1.1 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.

16.2.1.2 `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 qapi_ADC_Open() .
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.

16.2.1.3 `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 qapi_ADC_Open() .
in	<i>Input_Prop_Ptr</i>	Properties pointer of channel provided by qapi_ADC_Get_Input_Properties() .
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.

16.2.1.4 `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 qapi_ADC_Open() .
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.

16.2.1.5 `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 qapi_ADC_Open() .
in	<i>In_Properties_Ptr</i>	Properties pointer of the channel provided by qapi_ADC_TM_Get_Input_Properties() .
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.

16.2.1.6 `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:currentreading<=*Threshold_Desired_Ptr
- ADC_TM_THRESHOLD_HIGHER:currentreading>=*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 qapi_ADC_Open() .
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.

16.2.1.7 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 the 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 qapi_ADC_Open() .
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.

16.2.1.8 qapi_Status_t qapi_ADC_TM_Set_Tolerance (qapi_ADC_Handle_t Handle, const qapi_Adcm_TM_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 <= original_value - tolerance
- Upper: The event triggers when the current_value >= original_value + tolerance

Parameters

in	<i>Handle</i>	Handle provided by qapi_ADC_Open() .
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.

16.2.1.9 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 qapi_ADC_Open() .
in	<i>keep_enabled</i>	Reserved parameter.

Returns

- QAPI_OK – Call succeeded.
- QAPI_ERROR – Call failed.
- QAPI_ERR_INVALID_PARAM – Invalid parameters were specified.

16.3 TSENS Data Types

This section provides the type definitions for temperature sensor APIs.

16.3.1 Data Structure Documentation

16.3.1.1 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.

16.3.1.2 struct `qapi_TSENS_Result_t`

TSENS temperature result structure.

Data fields

Type	Parameter	Description
int	deg_c	Temperature in degrees Celsius.

16.3.2 Typedef Documentation

16.3.2.1 typedef `void(* QAPI_Tsens_Threshold_Cb_Type)(void *pCtxt, const qapi_TSENS_CallbackPayloadType_t *pPayload)`

TSENS callback function type.

```
typedef void* qapi_TSENS_Handle_t
```

TSENS handler type.

16.3.3 Enumeration Type Documentation

16.3.3.1 enum qapi_TSENS_ThresholdType_t

Enumeration of TSENS temperature thresholds.

Enumerator:

<i>QAPIS_TSENS_THRESHOLD_LOWER</i>	Lower threshold.
<i>QAPIS_TSENS_THRESHOLD_UPPER</i>	Upper threshold.
<i>QAPIS_TSENS_NUM_THRESHOLDS</i>	Number of thresholds.

16.4 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 demonstrates usage of this interface. The example
- * below opens TSENS to obtain a handle, gets the number of sensors, sets
- * temperature 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;
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;

```

16.4.1 Function Documentation

16.4.1.1 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.

16.4.1.2 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 qapi_TSENS_Open() .
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.

16.4.1.3 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 qapi_TSENS_Open() .
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.

16.4.1.4 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 qapi_TSENS_Open() .
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.

16.4.1.5 `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 reenable them if they were previously disabled by a call to [qapi_TSENS_Set_Enable_Thresholds\(\)](#).

Parameters

in	<i>Handle</i>	Handle provided by qapi_TSENS_Open() .
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.

16.4.1.6 `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 qapi_TSENS_Open() .
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.

16.4.1.7 qapi_Status_t qapi_TSENS_Close(qapi_TSENS_Handle_t *Handle*)

Closes TSENS.

Parameters

in	<i>Handle</i>	Handle provided by qapi_TSENS_Open() .
-----------	---------------	--

Returns

- QAPI_OK – Call succeeded.
- QAPI_ERROR – Call failed.
- QAPI_ERR_INVALID_PARAM – Invalid parameters were specified.

17 ALI IoT Cloud

This chapter describes the iot data types and APIs.

- [IoT debug Data Types](#)
- [IoT debug APIs](#)
- [IoT mqtt Data Types](#)
- [IoT mqtt APIs](#)
- [IoT coap Data Types](#)
- [IoT coap APIs](#)

17.1 IoT debug Data Types

17.1.1 Enumeration Type Documentation

17.1.1.1 enum IOT_LogLevel

```
IOT_LOG_EMERG
IOT_LOG_CRIT,
IOT_LOG_ERROR,
IOT_LOG_WARNING,
IOT_LOG_INFO,
IOT_LOG_DEBUG,
```

17.2 IoT debug APIs

17.2.1 Function Documentation

17.2.1.1 void IOT_OpenLog(const char *ident)

Start printing log information (log), accept a const char * for the entry, said module name

Parameters

in	ident	Specify the module name
----	-------	-------------------------

Returns

None.

17.2.1.2 void IOT_CloseLog(void)

Stop printing log information (log), into the Senate is empty

Parameters

Returns

None.

17.2.1.3 void IOT_SetLogLevel(IOT_LogLevel level)

Set the print log level, accept the entry from 1 to 5, the greater the number, the more detailed printing

Parameters

in	level	Specify the log level
----	-------	-----------------------

Returns

None.

17.2.1.4 void IOT_DumpMemoryStats(IOT_LogLevel level)

Debugging function, the use of print memory statistics, into the parameters 1-5, the greater the number, the more detailed printing

Parameters

in	level	Specify the log level
----	-------	-----------------------

Returns

None.

17.3 IoT mqtt Data Types

17.3.1 Enumeration Type Documentation

17.3.1.1 enum iotx_mqtt_qos_t

Enumerator:

IOTX_MQTT_QOS0
IOTX_MQTT_QOS1
IOTX_MQTT_QOS2

17.3.1.2 enum iotx_mqtt_event_type_t

Enumerator:

IOTX_MQTT_EVENT_UNDEF Undefined event
IOTX_MQTT_EVENT_DISCONNECT MQTT disconnect event
IOTX_MQTT_EVENT_RECONNECT MQTT reconnect event
IOTX_MQTT_EVENT_SUBSCRIBE_SUCCESS A ACK to the specific subscribe which specify by packet-id be received
IOTX_MQTT_EVENT_SUBSCRIBE_TIMEOUT No ACK to the specific subscribe which specify by packet-id be received in timeout period
IOTX_MQTT_EVENT_SUBSCRIBE_NACK A failed ACK to the specific subscribe which specify by packet-id be received
IOTX_MQTT_EVENT_UNSUBSCRIBE_SUCCESS A ACK to the specific unsubscribe which specify by packet-id be received
IOTX_MQTT_EVENT_UNSUBSCRIBE_TIMEOUT No ACK to the specific unsubscribe which specify by packet-id be received in timeout period
IOTX_MQTT_EVENT_UNSUBSCRIBE_NACK A failed ACK to the specific unsubscribe which specify by packet-id be received
IOTX_MQTT_EVENT_PUBLISH_SUCCESS A ACK to the specific publish which specify by packet-id be received
IOTX_MQTT_EVENT_PUBLISH_TIMEOUT No ACK to the specific publish which specify by packet-id be received in timeout period

IOTX_MQTT_EVENT_PUBLISH_NACK A failed ACK to the specific publish which specify by packet-id be received

IOTX_MQTT_EVENT_PUBLISH_RECVEIVED MQTT packet published from MQTT remote broker be received

17.3.2 Typedef Documentation

17.3.2.1 typedef void (*iotx_mqtt_event_handle_func_fpt)(void *pcontext, void *pclient, iotx_mqtt_event_msg_pt msg);

It define a datatype of function pointer. This type of function will be called when a related event occur.

Parameters

out	<i>pcontext</i>	the program context
out	<i>pclient</i>	the MQTT client.
out	<i>msg</i>	the event message

Returns

None.

17.3.3 Data Structure Documentation

17.3.3.1 struct iotx_mqtt_topic_info_t

Data fields

T	Parameter	Descr
uint16_t	packet_id	Specify the packet id
uint8_t	qos	Specify the qos
uint8_t	dup	Specify the dup
uint8_t	retain	Specify the retain
uint16_t	topic_len	Specify the topic len
uint16_t	payload_len	Specify the payload len
const char *	ptopic	Specify the ptopic
const char *	payload	Specify the payload

17.3.3.2 struct iotx_mqtt_event_msg_t

Data fields

T	Parameter	Descr
iotx_mqtt_event_type_t	event_type	Specify the event type

void *	msg	<p>Specify the detail event information. @msg means different to different event types:</p> <ol style="list-style-type: none"> 1) IOTX_MQTT_EVENT_UNKNOWN, IOTX_MQTT_EVENT_DISCONNECT, IOTX_MQTT_EVENT_RECONNECT : Its data type is string and the value is detail information. 2) IOTX_MQTT_EVENT_SUBSCRIBE_SUCCESS, IOTX_MQTT_EVENT_SUBSCRIBE_TIMEOUT, IOTX_MQTT_EVENT_SUBSCRIBE_NACK, IOTX_MQTT_EVENT_UNSUBSCRIBE_SUCCESS, IOTX_MQTT_EVENT_UNSUBSCRIBE_TIMEOUT, IOTX_MQTT_EVENT_UNSUBSCRIBE_NACK IOTX_MQTT_EVENT_PUBLISH_SUCCESS, IOTX_MQTT_EVENT_PUBLISH_TIMEOUT, IOTX_MQTT_EVENT_PUBLISH_NACK : Its data type is @uint32_t and the value is MQTT packet identifier. 3) IOTX_MQTT_EVENT_PUBLISH_RECEIVED:
--------	-----	--

17.3.3.3 struct iotx_mqtt_event_handle_t

The structure of MQTT event handle

Data fields

T	Parameter	Descript
iotx_mqtt_event_handle_func_t	h_fp	
void*	pcontext	

17.3.3.4 struct iotx_mqtt_param_t

The structure of MQTT initial parameter

T	Parameter	Descr
uint16_t	port	Specify MQTT broker port
const char *	host	Specify MQTT broker host
const char*	client_id	Specify MQTT connection client id
const char*	username	Specify MQTT user name
const char*	password	Specify MQTT password
const char*	pub_key	Specify MQTT transport channel and key. If the value is NULL, it means that use TCP channel, If the value is NOT NULL, it means that use SSL/TLS channel and @pub_key point to the CA certification
uint8_t	clean_session	Specify MQTT clean session or not
uint32_t	request_timeout	Specify timeout of a MQTT request in millisecond
uint32_t	keepalive_interval_ms	Specify MQTT keep-alive interval in millisecond
char*	pwrite_buf	Specify write-buffer
uint32_t	write_buf_size	Specify size of write-buffer in byte

char*	pread_buf	Specify read-buffer
uint32_t	read_buf_size	Specify size of read-buffer in byte
iotx_mqtt_event_handle_t	handle_event	Specify MQTT event handle

17.4 IoT mqtt APIs

17.4.1 Function Documentation

17.4.1.1 int IOT_SetupConnInfo(const char *product_key,const char *device_name,const char *device_secret,void **info_ptr);

MQTT connection preparation, user name and password for MQTT connection based on DeviceName + DeviceSecret + ProductKey

Parameters

in	product_key	specify the product key
in	device_name	specify the device name
in	device_secret	specify the device secret
out	info_ptr	specify the connect info

Returns

0 success
not 0 failed

17.4.1.2 void *IOT_MQTT_Construct(iotx_mqtt_param_t *pInitParams);

Construct the MQTT client This function initialize the data structures, establish MQTT connection.

Parameters

in	<i>pInitParams</i>	Specify the MQTT client parameter.
----	--------------------	------------------------------------

Returns

- NULL construct failed;
- NOT NULL, the handle of MQTT client.

17.4.1.3 int IOT_MQTT_Destroy(void **phandle);

Deconstruct the MQTT client

This function disconnect MQTT connection and release the related resource.

Parameters

in	<i>phandle</i>	pointer of handle, specify the MQTT client
----	----------------	--

Returns

- 0, deconstruct success
- -1, deconstruct failed.

17.4.1.4 int IOT_MQTT_Yield(void *handle, int timeout_ms);

Handle MQTT packet from remote server and process timeout request
which include the MQTT subscribe, unsubscribe, publish(QOS >= 1), reconnect, etc..

Parameters

in	<i>handle</i>	specify the MQTT client.
in	<i>timeout</i>	specify the timeout in millisecond in this loop.

Returns

17.4.1.5 int IOT_MQTT_CheckStateNormal(void *handle);

check whether MQTT connection is established or not

Parameters

in	<i>handle</i>	specify the MQTT client.
-----------	---------------	--------------------------

Returns

- true, MQTT in normal state
- false, MQTT in abnormal state.

17.4.1.6 int IOT_MQTT_Subscribe(void *handle, const char *topic_filter, iotx_mqtt_qos_t qos, iotx_mqtt_event_handle_func_fpt topic_handle_func, void *pcontext);

Subscribe MQTT topic.

Parameters

in	<i>handle</i>	specify the MQTT client.
in	<i>topic_filter</i>	specify the topic filter
in	<i>qos</i>	specify the MQTT Requested QoS
in	<i>topic_handle_func</i>	specify the topic handle callback-function
in	<i>pcontext</i>	specify context. When call @topic_handle_func, it will be passed back

Returns

- -1, subscribe failed.
- >=0, subscribe successful.
- The value is a unique ID of this request.
- The ID will be passed back when callback @iotx_mqtt_param_t:handle_event.

*/

17.4.1.7 int IOT_MQTT_Unsubscribe(void *handle, const char *topic_filter);

Unsubscribe MQTT topic.

Parameters

in	<i>handle</i>	specify the MQTT client.
in	<i>topic_filter</i>	specify the topic filter.

Returns

- -1, unsubscribe failed.
- >=0, unsubscribe successful.
- The value is a unique ID of this request.

- The ID will be passed back when callback @iotx_mqtt_param_t:handle_event.

17.4.1.8 int IOT_MQTT_Publish(void *handle, const char *topic_name, iotx_mqtt_topic_info_pt topic_msg);

Publish message to specific topic.

Parameters

in	<i>handle</i>	specify the MQTT client.
in	<i>topic_name</i>	specify the topic name
in	<i>topic_msg</i>	specify the topic message

Returns

- -1, publish failed.
- 0, publish successful, where QoS is 0.
- >0, publish successful, where QoS is >= 0.
- The value is a unique ID of this request.
- The ID will be passed back when callback @iotx_mqtt_param_t:handle_event.

17.5 IoT coap Data Types

17.5.1 Enumeration Type Documentation

17.5.1.1 enum iotx_ret_code_t

Enumerator:

<i>IOTX_ERR_MSG_TOO_LOOG</i>	The payload too loog
<i>IOTX_ERR_URI_TOO_LOOG</i>	URI length too long
<i>IOTX_ERR_NOT_AUTHED</i>	Client isn't authed
<i>IOTX_ERR_AUTH_FAILED</i>	Client authed failed
<i>IOTX_ERR_BUFF_TOO_SHORT</i>	Buffer too short
<i>IOTX_ERR_NO_MEM</i>	Malloc failed
<i>IOTX_ERR_INVALID_PARAM</i>	Invalid parameter
<i>IOTX_SUCCESS</i>	Success

17.5.1.2 enum otx_content_type_t

The message payload encode format

Enumerator:

IOTX_CONTENT_TYPE_JSON
IOTX_CONTENT_TYPE_CBOR

17.5.1.3 enum iotx_msg_type_t

The message type

<i>IOTX_MESSAGE_CON</i>	confirmable message
<i>IOTX_MESSAGE_NON</i>	non-confirmable message

17.5.1.4 enum iotx_coap_event_t

IoTx events to notify application

Enumerator:

IOTX_COAP_EVENT_SEND_FAILED
IOTX_COAP_EVENT_RECV_FAILED
IOTX_COAP_EVENT_AUTH_FAILED

17.5.1.5 enum iotx_coap_resp_code_t

Enumerator:

<i>IOXT_COAP_RESP_CODE_CONTENT</i>	Mapping to 2.05, Content
<i>IOXT_COAP_RESP_CODE_BAD_REQUEST</i>	Mapping to 4.00, Bad Request
<i>IOXT_COAP_RESP_CODE_UNAUTHORIZED</i>	Mapping to 4.01, Token is invalid or expire
<i>IOXT_COAP_RESP_CODE_NOT_FOUND</i>	Mapping to 4.04, Path or uri is not found
<i>IOXT_COAP_RESP_CODE_URL_TOO_LONG</i>	Mapping to 4.14, The request url is too long
<i>IOXT_COAP_RESP_CODE_INTERNAL_SERVER_ERROR</i>	Mapping to 5.00, Internal server error

17.5.2 Typedef Documentation

17.5.2.1 typedef void (*iotx_event_handle_t)(void *context, iotx_coap_event_t event, void *p_data);

Callback function to notify the application events

17.5.2.2 typedef void (*iotx_response_callback_t)(void *p_arg, void *p_message);

Callback function to handle the response message

17.5.2.3 struct otx_deviceinfo_t

Type	Parameter	Descript
char[IOTX_PRODUCT_KEY_LEN + 1]	product_key	specify the product key
char[IOTX_DEVICE_NAME_LEN + 1]	device_name	specify the device name
char[IOTX_DEVICE_ID_LEN + 1]	device_id	specify the device id
char[IOTX_DEVICE_SECRET_LEN + 1]	device_secret	specify the device secret

17.5.2.4 struct iotx_coap_config_t

IoTx initialize parameters

Type	Parameter	Descript
char*	p_url	specify the url
iotx_deviceinfo_t*	p_devinfo	specify the device info
iotx_event_handle_t	event_handle	Not use

17.5.2.5 struct iotx_message_t

IoTx message definition

Type	Parameter	Descript
unsigned char *	p_payload	specify the message payload .
unsigned short	payload_len	specify the message payload len
iotx_content_type_t	content_type	specify the content type
iotx_msg_type_t	msg_type	specify the message type

void	user_data	Not use
iotx_response_callback_t	resp_callback	specify the response callback

17.6 IoT coap APIs

17.6.1 Function Documentation

17.6.1.1 iotx_coap_context_t *IOT_CoAP_Init(iotx_coap_config_t *p_config);

The CoAP instance constructor, which is the iotx_coap_config_t structure.

Parameters

in	p_config	specify the MQTT client.
----	----------	--------------------------

Returns

the created CoAP session handle

17.6.1.2 void IOT_CoAP_Deinit(iotx_coap_context_t **p_context);

CoAP instance of the destruction function, into the entry IOT_CoAP_Init () created by the handle

Parameters

in	p_context	specify the MQTT client.
----	-----------	--------------------------

Returns

None

17.6.1.3 int IOT_CoAP_DeviceNameAuth(iotx_coap_context_t *p_context);

DeviceName, DeviceSecret, and ProductKey are device authentication based on the console application

Parameters

in	p_context	specify the MQTT client.
----	-----------	--------------------------

Returns

The iotx_ret_code_t type result

17.6.1.4 int IOT_CoAP_Yield(iotx_coap_context_t *p_context);

CoAP session phase, the call has been successfully established after the call, check and receive the server to the CoAP Request reply message

Parameters

in	p_context	specify the MQTT client.
----	-----------	--------------------------

Returns

The iotx_ret_code_t type result

17.6.1.5 int IOT_CoAP_SendMessage(iotx_coap_context_t *p_context, char *p_path, iotx_message_t *p_message);

CoAP session phase, the connection has been successfully established after the call, organize a complete CoAP message sent to the server

Parameters

in	p_context	specify the MQTT client.
in	p_path	specify the topic name
in	p_message	specify the topic message

Returns

The iotx_ret_code_t type result

17.6.1.6 int IOT_CoAP_GetMessagePayload(void *p_message, unsigned char **pp_payload, int *p_len);

The CoAP session phase obtains the packet load from the server's CoAP Response packe

Parameters

in	p_message	specify the MQTT client.
in	pp_payload	specify the topic name
in	p_len	specify the topic message

Returns

The iotx_ret_code_t type result

17.6.1.7 int IOT_CoAP_GetMessageCode(void *p_message, iotx_coap_resp_code_t *p_resp_code);

In the CoAP session phase, obtain the Respond Code from the server's CoAP Response message

Parameters

in	p_message	specify the MQTT client.
in	p_resp_code	specify the topic name

Returns

The iotx_ret_code_t type result

18 AT Engine

18.1 AT Engine APIs

18.1.1 Function Documentation

18.1.1.1 unsigned short qapi_DAM_Visual_AT_Input (const unsigned char *data, unsigned short length);
Write data to visual UART port.

Parameters

in	data	AT command
in	length	Length of AT command

Note: The AT command need “\r” to end.

Returns

<0 error
>0 data len

18.1.1.2 void qapi_DAM_Visual_AT_Output (unsigned char *data, unsigned short length);
Read data from visual UART port.

Parameters

out	data	Response of AT command or URC
in	length	Response length of AT command or URC

Returns

18.1.1.3 void qapi_DAM_Update_App (void)

Update customer app with below two necessary conditions.

1. /custapp/cust_app.update must be existed.
2. /custapp/cust_app.update is legal DAM program.

19 Device Information Module

This chapter describes the device information data types and APIs.

- [DeviceInformation](#)

19.1 Device Information

19.1.1 Define Documentation

19.1.1.1 #define QAPI_DEVICE_INFO_BUF_SIZE 128

Maximum size of [qapi_Device_Info_t](#) valuebuf.

19.1.2 Data Structure Documentation

19.1.2.1 struct qapi_Device_Info_t

QAPI device information structure.

Data fields

Type	Parameter	Description
qapi_Device_Info_ID_t	id	Required information ID.
qapi_Device_Info_Type_t	info_type	Response type.
union qapi_Device_Info_t	u	

19.1.2.2 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.

19.1.2.3 struct qapi_Device_Info_t.u.valuebuf

Data fields

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

19.1.3 Enumeration Type Documentation

19.1.3.1 enum qapi_Device_Info_ID_t

Device information types.

Enumerator:

<i>QAPI_DEVICE_INFO_BUILD_ID_E</i>	Device BUILD_ID.
<i>QAPI_DEVICE_INFO_IMEI_E</i>	Device IMEI.
<i>QAPI_DEVICE_INFO_IMSI_E</i>	UIM IMSI.
<i>QAPI_DEVICE_INFO_OS_VERSION_E</i>	Device OS version.
<i>QAPI_DEVICE_INFO_MANUFACTURER_E</i>	Device manufacturer.
<i>QAPI_DEVICE_INFO_MODEL_ID_E</i>	Device model ID.
<i>QAPI_DEVICE_INFO_BATTERY_STATUS_E</i>	Device battery status.
<i>QAPI_DEVICE_INFO_BATTERY_PERCENTAGE_E</i>	Device battery percentage.
<i>QAPI_DEVICE_INFO_TIME_ZONE_E</i>	Device time zone.
<i>QAPI_DEVICE_INFO_ICCID_E</i>	Device ICCID.
<i>QAPI_DEVICE_INFO_4G_SIG_STRENGTH_E</i>	Network signal strength.
<i>QAPI_DEVICE_INFO_BASE_STATION_ID_E</i>	Network base station ID.
<i>QAPI_DEVICE_INFO_MCC_E</i>	Network MCC.
<i>QAPI_DEVICE_INFO_MNC_E</i>	Network MNC.
<i>QAPI_DEVICE_INFO_SERVICE_STATE_E</i>	Network service status.
<i>QAPI_DEVICE_INFO_MBN_E</i>	Device MBN.
<i>QAPI_DEVICE_INFO_TAC_E</i>	Network tracking area code.
<i>QAPI_DEVICE_INFO_CELL_ID_E</i>	Network cell ID.
<i>QAPI_DEVICE_INFO_RCCS_E</i>	Network RRC state.
<i>QAPI_DEVICE_INFO_EMMS_E</i>	Network EMM state.
<i>QAPI_DEVICE_INFO_EMMSUBS_E</i>	Network EMM substate.
<i>QAPI_DEVICE_INFO_SERVING_RSRQ_E</i>	Serving cell RSRQ.
<i>QAPI_DEVICE_INFO_NEIGHBOUR_RSRP_E</i>	Neighbor cell RSRP.
<i>QAPI_DEVICE_INFO_NEIGHBOUR_RSRQ_E</i>	Neighbor cell RSRQ.
<i>QAPI_DEVICE_INFO_NEIGHBOUR_PCI_E</i>	Neighbor cell PCI.
<i>QAPI_DEVICE_INFO_NEIGHBOUR_EARFCN_E</i>	Neighbor cell EARFCN.
<i>QAPI_DEVICE_INFO_TADV_E</i>	Device timing advance.
<i>QAPI_DEVICE_INFO_NETWORK_IND_E</i>	Network indication.

19.1.3.2 enum qapi_Device_Info_Type_t

Device information response types.

Enumerator:

<i>QAPI_DEVICE_INFO_TYPE_BOOLEAN_E</i>	Response type is Boolean.
<i>QAPI_DEVICE_INFO_TYPE_INTEGER_E</i>	Response type is integer.
<i>QAPI_DEVICE_INFO_TYPE_BUFFER_E</i>	Response type is buffer.

19.1.4 Function Documentation

19.1.4.1 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.

19.1.4.2 `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.

19.1.4.3 `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.

19.1.4.4 `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.

19.1.4.5 `qapi_Status_t qapi_Device_Info_Reset(void)`

Resets the device.

Returns

QAPI_OK on success,

QAPI_ERROR on failure.

19.2 **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.

20 Use Cases

This chapter provides a recommended API use case for the DSS APIs and an example of a socket API use case.

20.1 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_hndl()` 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

20.2 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.

20.2.1 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.

20.2.2 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.

20.3 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_H
    dl_t          sslCtx;
    qapi_Net_SSL_Con_Hd
    l_t          sslCon;
    qapi_Net_SSL_Confi
    g_t          config; qapi_Net_SSL_Role_t role;
} SSL_INST;

```

20.3.1 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;
}

```

20.3.2 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;
    }
}

```

20.3.3 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;
}

```

20.3.4 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_Velocity = 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;
}
```

20.3.5 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);
```

20.3.6 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);
```

20.3.7 TLS/DTLS Close Context Object

```
if (ssl.sslCtx)  
{  
    qapi_Net_SSL_Obj_Free(ssl.sslCtx);  
    ssl,sslCtx = QAPI_NET_SSL_INVALID_HANDLE;  
}
```

A References

Acronyms and Terms

Acronym or term	Definition
APN	Access point name
BSD	Berkely Software Distribution
CE	Callend
DNS	Domain name service
DSS	Data services sockets
MTU	Maximum transmission unit
netctrl	Net control
PDP	Packet Data Protocol
QAPI	SIMCom API
QMI	SIMCom messaging interface

Contact us:

Shanghai SIMCom wireless solutions Ltd.

Address: Building A, SIM Technology Building, No. 633 Jinzhong Road, Shanghai,
P. R. China 200335

Tel: +86 21 3252 3300

Fax: +86 21 3252 3020

URL: www.simcomm2m.com