Qualcomm Technologies, Inc.

Qualcomm Application Programming Interface for MDM9206 TX3.0

Interface Specification

80-P8101-32 Rev. B

May 4, 2018

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

Revision History

Revision	Date	Description
A	Jan 2018	Initial release
B	May 2018	Changed document title, removed use cases information. Updated Sections 4.1.1, 4.1.2, 5.1.1, 5.1.2.11, 17.1.1.1,17.1.1.7, 17.1.4.18, 17.1.4.19, 21.1.3.1, and 22.1. Added Sections 5.1.2.2, 5.1.2.9, 5.2.1.4, 5.10, 5.19, 17.1.1.2, 17.1.3.6, 20.2.1.3, and 24.2.

Contents

1 Introduction

1.1 Purpose

This document is the reference specification for the Qualcomm Application Programming Interface (QAPI) for the MDM9206 ThreadX (TX) OS version 3.0.

The QAPIs are designed to facilitate the development of IoT applications.

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

1.2 Conventions

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

1.3 Technical Assistance

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

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

2 Functional Overview

The QAPIs are designed to facilitate the development of IoT applications. Chapter 3 contains details on each of the QAPIs.

Figure 2-1 illustrates the high level IOT application architecture in ThreadX. The QAPIs enable the application to use the features provided by the HLOS and modem on the MDM.

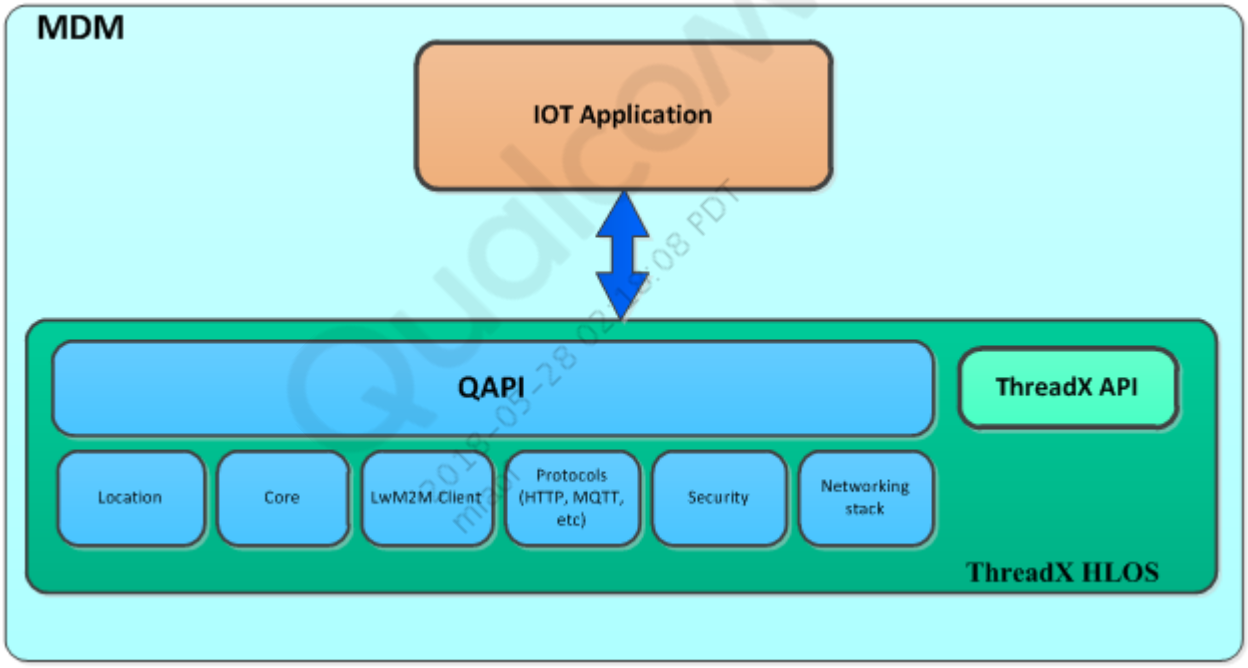

Figure 2-1 High level IOT application architecture in ThreadX

For use cases pertaining to these QAPIs, refer to *MDM9206 ThreadX QAPI Usage Guide* (80-P8101-35).

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
- Add Filters for an MO Exception IP Data Call
- Remove Filters for an MO Exception IP Data Call
- Send Non-IP UL Data

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

Extended Radio Technology

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

Supported Extended Radio Technologies

- #define **QAPI_DSS_EXT_RADIO_TECH_MAX** 1

MO Exception Data

- #define QAPI_DSS_MO_EXCEPTION_NONE 0x00
- #define QAPI_DSS_MO_EXCEPTION_IP_DATA 0x01
- #define QAPI_DSS_MO_EXCEPTION_NONIP_DATA 0x02

Call Information

- #define QAPI_DSS_CALL_INFO_USERNAME_MAX_LEN 127
- #define QAPI_DSS_CALL_INFO_PASSWORD_MAX_LEN 127
- #define QAPI_DSS_CALL_INFO_APN_MAX_LEN 150

Device Name

For example, rmnet_sdioxx, rmnet_xx, etc.

- #define QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN 12

Maximum Client Handles Supported

- #define **QAPI_DSS_MAX_DATA_CALLS** 20

QAPI_DSS Error Codes

- #define QAPI_DSS_SUCCESS 0
- #define QAPI_DSS_ERROR -1

IP Versions

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

Supported Modes of Operation

- #define **QAPI_DSS_MODE_GENERAL** 0

Maximum Supported MO Exception Filters

- #define **QAPI_DSS_MAX_EXCEPTION_FILTERS** 255

Maximum IPv6 Address Length

- #define **QAPI_DSS_IPV6_ADDR_LEN** 16

MO Exception Data Filter Error Mask

- typedef uint64_t **qapi_DSS_MO_Filter_Error_Mask_t**
- #define QAPI_DSS_FILTER_PARAM_NONE_V01 0x00000000
- #define QAPI_DSS_FILTER_PARAM_IP_VERSION_V01 0x00000001
- #define QAPI_DSS_FILTER_PARAM_IPV4_SRC_ADDR_V01 0x00000002
- #define QAPI_DSS_FILTER_PARAM_IPV4_DEST_ADDR_V01 0x00000004
- #define QAPI_DSS_FILTER_PARAM_IPV4_TOS_V01 0x00000008
- #define QAPI_DSS_FILTER_PARAM_IPV6_SRC_ADDR_V01 0x00000010
- #define QAPI_DSS_FILTER_PARAM_IPV6_DEST_ADDR_V01 0x00000020
- #define QAPI_DSS_FILTER_PARAM_IPV6_TRF_CLS_V01 0x00000040
- #define QAPI_DSS_FILTER_PARAM_IPV6_FLOW_LABEL_V01 0x00000080

- #define QAPI_DSS_FILTER_PARAM_XPORT_PROT_V01 0x00000100
- #define QAPI_DSS_FILTER_PARAM_TCP_SRC_PORT_V01 0x00000200
- #define QAPI_DSS_FILTER_PARAM_TCP_DEST_PORT_V01 0x00000400
- #define QAPI_DSS_FILTER_PARAM_UDP_SRC_PORT_V01 0x00000800
- #define QAPI_DSS_FILTER_PARAM_UDP_DEST_PORT_V01 0x00001000
- #define QAPI_DSS_FILTER_PARAM_ICMP_TYPE_V01 0x00002000
- #define QAPI_DSS_FILTER_PARAM_ICMP_CODE_V01 0x00004000
- #define QAPI_DSS_FILTER_PARAM_ESP_SPI_V01 0x00008000
- #define QAPI_DSS_FILTER_PARAM_AH_SPI_V01 0x00010000

MO Exception Data IPv4 Filter Mask

- typedef uint64_t **qapi_DSS_IPv4_Filter_Mask_t**
- #define QAPI_DSS_IPV4_FILTER_MASK_NONE 0x00000000
- #define QAPI_DSS_IPV4_FILTER_MASK_SRC_ADDR 0x00000001
- #define QAPI_DSS_IPV4_FILTER_MASK_DEST_ADDR 0x00000002
- #define QAPI_DSS_IPV4_FILTER_MASK_TOS 0x00000004

MO Exception Data IPv6 Filter Mask

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

Transport Port Filter Mask Information

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

ICMP Filter Mask Information

- typedef uint64_t **qapi_DSS_ICMP_Info_Filter_Mask_t**
- #define QAPI_DSS_ICMP_FILTER_MASK_NONE 0x00000000
- #define QAPI_DSS_ICMP_FILTER_MASK_MSG_TYPE 0x00000001

- #define QAPI_DSS_ICMP_FILTER_MASK_MSG_CODE 0x00000002

IPSec Filter Mask Information

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

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_MO_EXCEPTION_NONE 0x00

None.

3.1.1.10 #define QAPI_DSS_MO_EXCEPTION_IP_DATA 0x01

MO exception IP data.

3.1.1.11 #define QAPI_DSS_MO_EXCEPTION_NONIP_DATA 0x02

MO exception non-IP data.

3.1.1.12 #define QAPI_DSS_CALL_INFO_USERNAME_MAX_LEN 127

Maxiumum length of the username.

3.1.1.13 #define QAPI_DSS_CALL_INFO_PASSWORD_MAX_LEN 127

Maxiumum length of the password.

3.1.1.14 #define QAPI_DSS_CALL_INFO_APN_MAX_LEN 150

Maxiumum length of the APN.

3.1.1.15 #define QAPI_DSS_CALL_INFO_DEVICE_NAME_MAX_LEN 12

Maxiumum length of the device name.

3.1.1.16 #define QAPI_DSS_SUCCESS 0

Indicates that the operation was successful.

3.1.1.17 #define QAPI_DSS_ERROR -1

Indicates that the operation was not successful.

3.1.1.18 #define QAPI_DSS_IP_VERSION_4 4

IP version v4.

3.1.1.19 #define QAPI_DSS_IP_VERSION_6 6

IP version v6.

3.1.1.20 #define QAPI_DSS_IP_VERSION_4_6 10

IP version v4v6.

3.1.1.21 #define QAPI_DSS_FILTER_PARAM_NONE_V01 0x00000000

No errors.

3.1.1.22 #define QAPI_DSS_FILTER_PARAM_IP_VERSION_V01 0x00000001

IP version.

3.1.1.23 #define QAPI_DSS_FILTER_PARAM_IPV4_SRC_ADDR_V01 0x00000002

IPv4 source address.

3.1.1.24 #define QAPI_DSS_FILTER_PARAM_IPV4_DEST_ADDR_V01 0x00000004

IPv4 destination address.

3.1.1.25 #define QAPI_DSS_FILTER_PARAM_IPV4_TOS_V01 0x00000008

IPv4 type of service.

3.1.1.26 #define QAPI_DSS_FILTER_PARAM_IPV6_SRC_ADDR_V01 0x00000010

IPv6 source address.

3.1.1.27 #define QAPI_DSS_FILTER_PARAM_IPV6_DEST_ADDR_V01 0x00000020

IPv6 destination address.

3.1.1.28 #define QAPI_DSS_FILTER_PARAM_IPV6_TRF_CLS_V01 0x00000040

IPv6 traffic class.

3.1.1.29 #define QAPI_DSS_FILTER_PARAM_IPV6_FLOW_LABEL_V01 0x00000080

IPv6 flow label.

3.1.1.30 #define QAPI_DSS_FILTER_PARAM_XPORT_PROT_V01 0x00000100

Transport protocol.

3.1.1.31 #define QAPI_DSS_FILTER_PARAM_TCP_SRC_PORT_V01 0x00000200

TCP source port.

3.1.1.32 #define QAPI_DSS_FILTER_PARAM_TCP_DEST_PORT_V01 0x00000400

TCP destination port.

3.1.1.33 #define QAPI_DSS_FILTER_PARAM_UDP_SRC_PORT_V01 0x00000800

UDP source port.

3.1.1.34 #define QAPI_DSS_FILTER_PARAM_UDP_DEST_PORT_V01 0x00001000

UDP destination port.

3.1.1.35 #define QAPI_DSS_FILTER_PARAM_ICMP_TYPE_V01 0x00002000

ICMP type.

3.1.1.36 #define QAPI_DSS_FILTER_PARAM_ICMP_CODE_V01 0x00004000

ICMP code.

3.1.1.37 #define QAPI_DSS_FILTER_PARAM_ESP_SPI_V01 0x00008000

Encapsulating SPI.

3.1.1.38 #define QAPI_DSS_FILTER_PARAM_AH_SPI_V01 0x00010000

Authentication header SPI.

3.1.1.39 #define QAPI_DSS_IPV4_FILTER_MASK_NONE 0x00000000

No parameters.

3.1.1.40 #define QAPI_DSS_IPV4_FILTER_MASK_SRC_ADDR 0x00000001

IPv4 source address.

3.1.1.41 #define QAPI_DSS_IPV4_FILTER_MASK_DEST_ADDR 0x00000002

IPv4 destination address.

3.1.1.42 #define QAPI_DSS_IPV4_FILTER_MASK_TOS 0x00000004

IPv4 traffic class.

3.1.1.43 #define QAPI_DSS_IPV6_FILTER_MASK_NONE 0x00000000

No parameters.

3.1.1.44 #define QAPI_DSS_IPV6_FILTER_MASK_SRC_ADDR 0x00000001

IPv6 source address.

3.1.1.45 #define QAPI_DSS_IPV6_FILTER_MASK_DEST_ADDR 0x00000002

IPv6 destination address.

3.1.1.46 #define QAPI_DSS_IPV6_FILTER_MASK_TRAFFIC_CLASS 0x00000004

IPv6 traffic class.

3.1.1.47 #define QAPI_DSS_IPV6_FILTER_MASK_FLOW_LABEL 0x00000008

IPv6 flow label.

3.1.1.48 #define QAPI_DSS_PORT_INFO_FILTER_MASK_NONE 0x00000000

No parameters.

3.1.1.49 #define QAPI_DSS_PORT_INFO_FILTER_MASK_SRC_PORT 0x00000001

Source port.

3.1.1.50 #define QAPI_DSS_PORT_INFO_FILTER_MASK_DEST_PORT 0x00000002

Destination port.

3.1.1.51 #define QAPI_DSS_ICMP_FILTER_MASK_NONE 0x00000000

No parameters.

3.1.1.52 #define QAPI_DSS_ICMP_FILTER_MASK_MSG_TYPE 0x00000001

Message type.

3.1.1.53 #define QAPI_DSS_ICMP_FILTER_MASK_MSG_CODE 0x00000002

Message code.

3.1.1.54 #define QAPI_DSS_IPSEC_FILTER_MASK_NONE 0x00000000

No parameters.

3.1.1.55 #define QAPI_DSS_IPSEC_FILTER_MASK_SPI 0x00000001

Security parameter index.

3.1.1.56 #define qapi_DSS_Pass_Pool_Ptr(*a, b*) dss_set_byte_pool(a,b)

Macro that passes a Byte Pool pointer for the DSS application.

Parameter a – Handle.

Parameter b – Pointer to the Byte Pool.

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

Note: This macro is only used in the DAM space.

3.1.1.57 #define qapi_DSS_Release(*a*) dss_release_byte_pool_release_handle(a)

Macro that releases a Byte Pool pointer for the DSS application.

Parameter a – Handle.

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

Note: This macro is only used in the DAM space.

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

Specifies call parameter values.

Data fields

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

3.1.2.3 struct qapi_DSS_Addr_s

Structure to represent the IP address.

Data fields

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

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.
long long	bytes_tx	Number of bytes transmitted.
long long	bytes_rx	Number of bytes received.

Type	Parameter	Description
unsigned long	pkts_dropped_-tx	Number of transmit packets dropped.
unsigned long	pkts_dropped_-rx	Number of receive packets dropped.

3.1.2.7 struct qapi_DSS_Evt_Payload_s

Event payload sent with event callbacks.

Data fields

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

3.1.2.8 struct qapi_DSS_IPv4_Filter_Address_Type_s

IPv4 address filter type.

Data fields

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

3.1.2.9 struct qapi_DSS_IPv4_Filter_TOS_Type_s

IPv4 TOS filter type.

Data fields

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

3.1.2.10 struct qapi_DSS_IPv4_Filter_Info_s

IPv4 filter rule information.

Data fields

Type	Parameter	Description
qapi_DSS_I-Pv4_Filter_-Mask_t	valid_params	Bitmask that denotes which parameters contain valid values.

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

3.1.2.11 struct qapi_DSS_IPv6_Filter_Address_Type_s

IPv6 address filter type.

Data fields

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

3.1.2.12 struct qapi_DSS_IPv6_Filter_Traffic_Type_s

IPv6 traffic class filter type.

Data fields

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

3.1.2.13 struct qapi_DSS_IPv6_Filter_Info_s

IPv6 filter rule information.

Data fields

Type	Parameter	Description
qapi_DSS_IPv6_Filter_Mask_t	valid_params	Bitmask that denotes which parameters contain valid values.
qapi_DSS_IPv6_Filter_Address_Type_t	src_addr	IPv6 source address.

Type	Parameter	Description
qapi_DSS_IPv6_Filter_Address_Type_t	dest_addr	IPv6 destination address.
qapi_DSS_IPv6_Filter_Traffic_Type_t	trf_cls	IPv6 traffic class.
uint32_t	flow_label	IPv6 flow label.

3.1.2.14 struct qapi_DSS_IP_Header_Filters_s

Internet protocol filter rule parameters.

Data fields

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

3.1.2.15 struct qapi_DSS_Port_Type_s

DSS port type.

Data fields

Type	Parameter	Description
uint16_t	port	Port.
uint16_t	range	Range.

3.1.2.16 struct qapi_DSS_Port_Filter_Info_s

TCP and UDP port filter rule parameters.

Data fields

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

Type	Parameter	Description
qapi_DSS_-Port_Type_t	src_port_info	Source port information.
qapi_DSS_-Port_Type_t	dest_port_info	Destination port information.

3.1.2.17 struct qapi_DSS_ICMP_Info_Filter_Type_s

ICMP filter rule parameters.

Data fields

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

3.1.2.18 struct qapi_DSS_IPSec_Info_Filter_Type_s

IPSec filter rule parameters.

Data fields

Type	Parameter	Description
qapi_DSS_I-PSec_Info_-Filter_Mask_t	valid_params	Bitmask that denotes which parameters contain valid values.
uint32_t	spi	Security parameter index for IPSec.

3.1.2.19 struct qapi_DSS_Xport_Header_Filters_s

Transport protocol filter rule parameters.

Data fields

Type	Parameter	Description
qapi_DSS_XP-ORT_Protocol-_t	xport_protocol	Depending on the value in xport_protocol, only one field of icmp_info, tcp_info, udp_info, esp_info, or ah_info is valid. QAPI_DSS_XPORT_PROTO_NONE implies that no transport level protocol parameters are valid.
qapi_DSS_-Port_Filter_-Info_t	tcp_info	Filter parameters for TCP.
qapi_DSS_-Port_Filter_-Info_t	udp_info	Filter parameters for UDP.

Type	Parameter	Description
qapi_DSS_I-CMP_Info_-Filter_Type_t	icmp_info	Filter parameters for ICMP.
qapi_DSS_I-PSec_Info_-Filter_Type_t	esp_info	Filter parameters for ESP.
qapi_DSS_I-PSec_Info_-Filter_Type_t	ah_info	Filter parameters for AH.

3.1.2.20 struct qapi_DSS_Filter_Rule_Type_s

MO exception data filter rules.

Data fields

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

3.1.2.21 struct qapi_DSS_Add_MO_Exception_Filters_Req_s

Add an MO exception data filters request.

Data fields

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

3.1.2.22 struct qapi_DSS_Add_MO_Exception_Filters_Rsp_s

Add an MO exception data filters response.

Data fields

Type	Parameter	Description
uint8_t	filter_handles_-valid	Set to TRUE if filter handles are being passed.

Type	Parameter	Description
uint32_t	filter_handles_-len	Set to the number of elements in the filter handles.
uint32_t	filter_handles	List of handles that uniquely identify added filter rules.
uint8_t	filter_rule_-error_valid	Set to TRUE if filter rule errors are being passed.
uint32_t	filter_rule_-error_len	Set to the number of elements in the filter rule error.
qapi_DSS_M-O_Filter_Error-_Mask_t	filter_rule_error	Error mask list for filter rule errors.

3.1.2.23 struct qapi_DSS_Remove_MO_Exceptional_Filters_s

Remove MO exception data filters.

Data fields

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

3.1.3 Typedef Documentation

3.1.3.1 typedef void(∗ qapi_DSS_Net_Ev_CB_t)(qapi_DSS_Hndl_t hndl,void ∗user_data,qapi_DSS_Net_Evt_t evt,qapi_DSS_Evt_Payload_t ∗payload_ptr)

Callback function prototype for DSS events.

Parameters

in	*hndl*	Handle to which this event is associated.
in	*user_data*	Application-provided user data.
in	*evt*	Event identifier.
in	*payload_ptr*	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.
QAPI_DSS_CALL_INFO_EXT_TECH_E Extended technology preference.
QAPI_DSS_CALL_INFO_MO_EXCEPTION_DATA_E MO exception data.

3.1.4.4 enum qapi_DSS_Net_Evt_e

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

Enumerator:

QAPI_DSS_EVT_INVALID_E Invalid event.
QAPI_DSS_EVT_NET_IS_CONN_E Call connected.
QAPI_DSS_EVT_NET_NO_NET_E Call disconnected.

QAPI_DSS_EVT_NET_RECONFIGURED_E Call reconfigured.
QAPI_DSS_EVT_NET_NEWADDR_E New address generated.
QAPI_DSS_EVT_NET_DELADDR_E Delete generated.
QAPI_DSS_EVT_NIPD_DL_DATA_E Non-IP downlink data.

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.
QAPI_DSS_NON_IP_FAMILY_E Non-IP family.

3.1.4.6 enum qapi_DSS_Data_Bearer_Tech_e

Bearer technology types.

Enumerator:

QAPI_DSS_DATA_BEARER_TECH_UNKNOWN_E Unknown bearer.
QAPI_DSS_DATA_BEARER_TECH_CDMA_1X_E 1X technology.
QAPI_DSS_DATA_BEARER_TECH_EVDO_REV0_E CDMA Rev 0.
QAPI_DSS_DATA_BEARER_TECH_EVDO_REVA_E CDMA Rev A.
QAPI_DSS_DATA_BEARER_TECH_EVDO_REVB_E CDMA Rev B.
QAPI_DSS_DATA_BEARER_TECH_EHRPD_E EHRPD.
QAPI_DSS_DATA_BEARER_TECH_FMC_E Fixed mobile convergence.
QAPI_DSS_DATA_BEARER_TECH_HRPD_E HRPD.
QAPI_DSS_DATA_BEARER_TECH_3GPP2_WLAN_E IWLAN.
QAPI_DSS_DATA_BEARER_TECH_WCDMA_E WCDMA.
QAPI_DSS_DATA_BEARER_TECH_GPRS_E GPRS.
QAPI_DSS_DATA_BEARER_TECH_HSDPA_E HSDPA.
QAPI_DSS_DATA_BEARER_TECH_HSUPA_E HSUPA.
QAPI_DSS_DATA_BEARER_TECH_EDGE_E EDGE.
QAPI_DSS_DATA_BEARER_TECH_LTE_E LTE.
QAPI_DSS_DATA_BEARER_TECH_HSDPA_PLUS_E HSDPA+.
QAPI_DSS_DATA_BEARER_TECH_DC_HSDPA_PLUS_E DC HSDPA+.
QAPI_DSS_DATA_BEARER_TECH_HSPA_E HSPA.
QAPI_DSS_DATA_BEARER_TECH_64_QAM_E 64 QAM.
QAPI_DSS_DATA_BEARER_TECH_TDSCDMA_E TD-SCDMA.
QAPI_DSS_DATA_BEARER_TECH_GSM_E GSM.
QAPI_DSS_DATA_BEARER_TECH_3GPP_WLAN_E IWLAN.

3.1.4.7 enum qapi_DSS_Call_Tech_Type_e

Call technology.

Enumerator:

QAPI_DSS_CALL_TECH_INVALID_E Invalid technology.

QAPI_DSS_CALL_TECH_CDMA_E CDMA.
QAPI_DSS_CALL_TECH_UMTS_E UMTS.

3.1.4.8 enum qapi_DSS_XPORT_Protocol_e

MO exception data transport protocol information.

Enumerator:

QAPI_DSS_XPORT_PROTO_NONE No transport protocol.
QAPI_DSS_XPORT_PROTO_ICMP Internet Control Messaging Protocol.
QAPI_DSS_XPORT_PROTO_TCP Transmission Control Protocol.
QAPI_DSS_XPORT_PROTO_UDP User Datagram Protocol.
QAPI_DSS_XPORT_PROTO_ESP Encapsulating Security Payload protocol.
QAPI_DSS_XPORT_PROTO_AH Authentication Header Protocol.
QAPI_DSS_XPORT_PROTO_ICMP6 ICMPv6 Protocol.
QAPI_DSS_XPORT_PROTO_TCPUDP TCP and UDP protocol; only applicable for remote socket requests.

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. This 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	*mode*	Mode of operation in which to initialize the library.

Returns

QAPI_OK – Initialization was successful.
QAPI_ERROR – Initialization failed.

Dependencies

None.

3.3 Release the DSS Netctrl Library

3.3.1 Function Documentation

3.3.1.1 qapi_Status_t qapi_DSS_Release (int *mode*)

Cleans up the DSS netctrl library. This 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	*mode*	Mode of operation in which to de-initialize the library.

Returns

QAPI_OK – Cleanup was successful.
QAPI_ERROR – Cleanup failed.

Dependencies

None.

3.4 Get the Data Service Handle

3.4.1 Function Documentation

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

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

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

Parameters

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

Returns

QAPI_OK – Operation was successful.
QAPI_ERROR – Operation failed.

Dependencies

qapi_DSS_Init() must have been called first.

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	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().

Returns

QAPI_OK – Operation was successful.
QAPI_ERROR – Operation failed.

Dependencies

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

3.6 Set the Data Call Parameter

3.6.1 Function Documentation

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

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

Parameters

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

Returns

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

Dependencies

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

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 notfications via a callback registered with qapi_DSS_Get_Data_Srvc_Hndl() indicating the data call bring-up status.

Parameters

in	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().

Returns

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

Dependencies

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

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_Srvc_Hndl() indicating the data call tear-down status.

Parameters

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

Returns

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

Dependencies

qapi_DSS_Init() must have been called first.
A valid handle must be obtained by qapi_DSS_Get_Data_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	*hndl*	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_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	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
out	*ce_reason*	Buffer to hold data call ending reason information.
in	*ip_family*	IP family for which the call end reason was requested.

Returns

QAPI_OK – Data call end reason was queried successfully.
QAPI_ERROR – Data call end reason query was unsuccessful.

Dependencies

qapi_DSS_Init() must have been called first.
A valid handle must be obtained by qapi_DSS_Get_Data_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_DSS_Hndl_t *hndl*, qapi_DSS_Call_Tech_Type_t * *call_tech*)

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

Parameters

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

Returns

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

Dependencies

qapi_DSS_Init() must have been called first.
A valid handle must be obtained by qapi_DSS_Get_Data_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_DSS_Hndl_t *hndl*, qapi_DSS_Data_Bearer_Tech_t * *bearer_tech*)

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

Parameters

in	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
in	*bearer_tech*	Pointer to where to retrieve the data bearer technology.

Returns

QAPI_OK – Data bearer technology was returned successfully.
QAPI_ERROR – Data bearer technology was not returned successfully.

Dependencies

qapi_DSS_Init() must have been called first.
A valid handle must be obtained by qapi_DSS_Get_Data_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	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
out	*buf*	Buffer to hold the data interface name string.
in	*len*	Length of the buffer allocated by the client.

Returns

QAPI_OK – Data interface name was returned successfully.
QAPI_ERROR – Data interface name was not returned successfully.

Dependencies

qapi_DSS_Init() must have been called first.
A valid handle must be obtained by qapi_DSS_Get_Data_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	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
out	*buf*	Buffer to hold the QMI port name string.
in	*len*	Length of the buffer allocated by the client.

Returns

QAPI_OK – Port name was returned successfully.
QAPI_ERROR – Port name was not returned successfully.

Dependencies

qapi_DSS_Init() must have been called first.
A valid handle must be obtained by qapi_DSS_Get_Data_Srvc_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	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
in	*ip_addr_cnt*	Pointer to where to retrieve the number of IP addresses associated with the DSS interface.

Returns

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

Dependencies

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

3.17 Get the IP Address Information

3.17.1 Function Documentation

3.17.1.1 qapi_Status_t qapi_DSS_Get_IP_Addr (qapi_DSS_Hndl_t *hndl*, qapi_DSS_-Addr_Info_t * *info_ptr*, int *len*)

Queries the IP address information structure (network order).

Parameters

in	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
out	*info_ptr*	Buffer containing the IP address information.
in	*len*	Number of IP address buffers

Returns

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

Dependencies

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

3.18 Get the IP Address Information Structure

3.18.1 Function Documentation

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

Queries the IP address information structure (network order).

Parameters

in	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
out	*info_ptr*	Buffer containing the IP address information.
in	*addr_family*	IPv4 / IPv6

Returns

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

Dependencies

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

3.19 Get the Link MTU Information

3.19.1 Function Documentation

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

Queries the MTU information associated with the link.

Parameters

in	*hndl*	Handle received from qapi_DSS_Get_Data_Srvc_Hndl().
out	*mtu*	Buffer containing the MTU information.

Returns

QAPI_OK – MTU query was successful.
QAPI_ERROR – MTU query was unsuccessful.

Dependencies

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

3.20 Add Filters for an MO Exception IP Data Call

3.20.1 Function Documentation

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

Adds filters for an MO exception IP data call.

Parameters

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

Returns

QAPI_OK – Adding filter rules was successful.
QAPI_ERROR – Adding filter rules was unsuccessful.

Dependencies

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

3.21 Remove Filters for an MO Exception IP Data Call

3.21.1 Function Documentation

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

Removes filters for an MO exception IP data call.

Parameters

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

Returns

QAPI_OK – Removing filter rules was successful.
QAPI_ERROR – Removing filter rules was unsuccessful.

Dependencies

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

3.22 Send Non-IP UL Data

3.22.1 Function Documentation

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

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

Parameters

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

Returns

QAPI_OK – Send Data was successful.
QAPI_ERROR – Send Data was unsuccessful.

Dependencies

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

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.

The BSD socket interface API is a collection of standard functions that allow the application to include Internet communications capabilities. 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 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.

BSD Socket Error Codes

- #define ENOBUFS 1
- #define ETIMEDOUT 2
- #define EISCONN 3
- #define EOPNOTSUPP 4
- #define ECONNABORTED 5
- #define EWOULDBLOCK 6
- #define ECONNREFUSED 7
- #define ECONNRESET 8
- #define EBADF 9
- #define EALREADY 10
- #define EMSGSIZE 12
- #define EPIPE 13
- #define EDESTADDRREQ 14
- #define ESHUTDOWN 15
- #define ENOPROTOOPT 16
- #define EHAVEOOB 17
- #define EADDRNOTAVAIL 19
- #define EADDRINUSE 20
- #define EAFNOSUPPORT 21
- #define EINPROGRESS 22
- #define ELOWER 23
- #define ENOTSOCK 24

- #define EIEIO 27
- #define ETOOMANYREFS 28
- #define EFAULT 29
- #define ENETUNREACH 30
- #define ENOTCONN 31

Socket Options

- #define SOL_SOCKET -1
- #define SOL_SOCKET -1
- #define SO_ACCEPTCONN 0x00002
- #define SO_REUSEADDR 0x00004
- #define SO_KEEPALIVE 0x00008
- #define SO_DONTROUTE 0x00010
- #define SO_BROADCAST 0x00020
- #define SO_USELOOPBACK 0x00040
- #define SO_LINGER 0x00080
- #define SO_OOBINLINE 0x00100
- #define SO_TCPSACK 0x00200
- #define SO_WINSCALE 0x00400
- #define SO_TIMESTAMP 0x00800
- #define SO_BIGCWND 0x01000
- #define SO_HDRINCL 0x02000
- #define SO_NOSLOWSTART 0x04000
- #define SO_FULLMSS 0x08000
- #define SO_SNDTIMEO 0x1005
- #define SO_RCVTIMEO 0x1006
- #define SO_ERROR 0x1007
- #define SO_RXDATA 0x1011
- #define SO_TXDATA 0x1012
- #define SO_MYADDR 0x1013
- #define SO_NBIO 0x1014
- #define SO_BIO 0x1015
- #define SO_NONBLOCK 0x1016
- #define SO_CALLBACK 0x1017

- #define SO_UDPCALLBACK 0x1019
- #define IPPROTO_IP 0
- #define IP_HDRINCL 2
- #define IP_MULTICAST_IF 9
- #define IP_MULTICAST_TTL 10
- #define IP_MULTICAST_LOOP 11
- #define IP_ADD_MEMBERSHIP 12
- #define IP_DROP_MEMBERSHIP 13
- #define IPV6_MULTICAST_IF 80
- #define IPV6_MULTICAST_HOPS 81
- #define IPV6_MULTICAST_LOOP 82
- #define IPV6_JOIN_GROUP 83
- #define IPV6_LEAVE_GROUP 84
- #define IP_EXCLUDE_LIST 17
- #define IP_OPTIONS 1
- #define IP_TOS 3
- #define IP_TTL_OPT 4
- #define IPV6_SCOPEID 14
- #define IPV6_UNICAST_HOPS 15
- #define IPV6_TCLASS 16

Flags for recv() and send()

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

4.1.1 Define Documentation

4.1.1.1 #define AF_UNSPEC 0

Address family is unspecified.

4.1.1.2 #define AF_INET 2

Address family is IPv4.

4.1.1.3 #define AF_INET6 3

Address family is IPv6.

4.1.1.4 #define AF_INET_DUAL46 4

Address family is IPv4 and IPv6.

4.1.1.5 #define SOCK_STREAM 1

Socket stream (TCP).

4.1.1.6 #define SOCK_DGRAM 2

Socket datagram (UDP).

4.1.1.7 #define SOCK_RAW 3

Raw socket.

4.1.1.8 #define ENOBUFS 1

No buffer space is available.

4.1.1.9 #define ETIMEDOUT 2

Operation timed out.

4.1.1.10 #define EISCONN 3

Socket is already connected.

4.1.1.11 #define EOPNOTSUPP 4

Operation is not supported.

4.1.1.12 #define ECONNABORTED 5

Software caused a connection abort.

4.1.1.13 #define EWOULDBLOCK 6

Socket is marked nonblocking and the requested operation will block.

4.1.1.14 #define ECONNREFUSED 7

Connection was refused.

4.1.1.15 #define ECONNRESET 8

Connection was reset by peer.

4.1.1.16 #define EBADF 9

An invalid descriptor was specified.

4.1.1.17 #define EALREADY 10

Operation is already in progress.

4.1.1.18 #define EMSGSIZE 12

Message is too long.

4.1.1.19 #define EPIPE 13

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

4.1.1.20 #define EDESTADDRREQ 14

Destination address is required.

4.1.1.21 #define ESHUTDOWN 15

Cannot send after a socket shutdown.

4.1.1.22 #define ENOPROTOOPT 16

Protocol is not available.

4.1.1.23 #define EHAVEOOB 17

Out of band.

4.1.1.24 #define EADDRNOTAVAIL 19

Cannot assign the requested address.

4.1.1.25 #define EADDRINUSE 20

Address is already in use.

4.1.1.26 #define EAFNOSUPPORT 21

Address family is not supported by the protocol family.

4.1.1.27 #define EINPROGRESS 22

Operation is in progress.

4.1.1.28 #define ELOWER 23

Lower layer (IP) error.

4.1.1.29 #define ENOTSOCK 24

Socket operation on nonsocket.

4.1.1.30 #define EIEIO 27

I/O error.

4.1.1.31 #define ETOOMANYREFS 28

Too many references.

4.1.1.32 #define EFAULT 29

Bad address.

4.1.1.33 #define ENETUNREACH 30

Network is unreachable.

4.1.1.34 #define ENOTCONN 31

Socket is not connected.

4.1.1.35 #define SOL_SOCKET -1

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

4.1.1.36 #define SOL_SOCKET -1

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

4.1.1.37 #define SO_ACCEPTCONN 0x00002

Socket has had listen().

4.1.1.38 #define SO_REUSEADDR 0x00004

Allow local address reuse.

4.1.1.39 #define SO_KEEPALIVE 0x00008

Keep connections alive.

4.1.1.40 #define SO_DONTROUTE 0x00010

Not used.

4.1.1.41 #define SO_BROADCAST 0x00020

Not used.

4.1.1.42 #define SO_USELOOPBACK 0x00040

Not used.

4.1.1.43 #define SO_LINGER 0x00080

Linger on close if data is present.

4.1.1.44 #define SO_OOBINLINE 0x00100

Leave the received OOB data in line.

4.1.1.45 #define SO_TCPSACK 0x00200

Allow TCP SACK (selective acknowledgment).

4.1.1.46 #define SO_WINSCALE 0x00400

Set the scaling window option.

4.1.1.47 #define SO_TIMESTAMP 0x00800

Set the TCP timestamp option.

4.1.1.48 #define SO_BIGCWND 0x01000

Large initial TCP congenstion window.

4.1.1.49 #define SO_HDRINCL 0x02000

User access to IP header for SOCK_RAW.

4.1.1.50 #define SO_NOSLOWSTART 0x04000

Suppress slowstart on this socket.

4.1.1.51 #define SO_FULLMSS 0x08000

Not used.

4.1.1.52 #define SO_SNDTIMEO 0x1005

Send a timeout.

4.1.1.53 #define SO_RCVTIMEO 0x1006

Receive a timeout.

4.1.1.54 #define SO_ERROR 0x1007

Socket error.

4.1.1.55 #define SO_RXDATA 0x1011

Get a count of bytes in sb_rcv.

4.1.1.56 #define SO_TXDATA 0x1012

Get a count of bytes in sb_snd.

4.1.1.57 #define SO_MYADDR 0x1013

Return my IP address.

4.1.1.58 #define SO_NBIO 0x1014

Set socket to Nonblocking mode.

4.1.1.59 #define SO_BIO 0x1015

Set socket to Blocking mode.

4.1.1.60 #define SO_NONBLOCK 0x1016

Set/get blocking mode via the optval parameter.

4.1.1.61 #define SO_CALLBACK 0x1017

Set/get the TCP zero_copy callback routine.

4.1.1.62 #define SO_UDPCALLBACK 0x1019

Set/get the UDP zero_copy callback routine.

4.1.1.63 #define IPPROTO_IP 0

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

4.1.1.64 #define IP_HDRINCL 2

IP header is included with the data.

4.1.1.65 #define IP_MULTICAST_IF 9

Set/get the IP multicast interface.

4.1.1.66 #define IP_MULTICAST_TTL 10

Set/get the IP multicast TTL.

4.1.1.67 #define IP_MULTICAST_LOOP 11

Set/get the IP multicast loopback.

4.1.1.68 #define IP_ADD_MEMBERSHIP 12

Add an IPv4 group membership.

4.1.1.69 #define IP_DROP_MEMBERSHIP 13

Drop an IPv4 group membership.

4.1.1.70 #define IPV6_MULTICAST_IF 80

Set the egress interface for multicast traffic.

4.1.1.71 #define IPV6_MULTICAST_HOPS 81

Set the number of hops.

4.1.1.72 #define IPV6_MULTICAST_LOOP 82

Enable/disable loopback for multicast.

4.1.1.73 #define IPV6_JOIN_GROUP 83

Join an IPv6 MC group.

4.1.1.74 #define IPV6_LEAVE_GROUP 84

Leave an IPv6 MC group.

4.1.1.75 #define IP_EXCLUDE_LIST 17

Set/get the exclude list for 255 RAW socket.

4.1.1.76 #define IP_OPTIONS 1

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

4.1.1.77 #define IP_TOS 3

IPv4 type of service and precedence.

4.1.1.78 #define IP_TTL_OPT 4

IPv4 time to live.

4.1.1.79 #define IPV6_SCOPEID 14

IPv6 IF scope ID.

4.1.1.80 #define IPV6_UNICAST_HOPS 15

IPv6 hop limit.

4.1.1.81 #define IPV6_TCLASS 16

IPv6 traffic class.

4.1.1.82 #define MSG_OOB 0x1

Send/receive out-of-band data.

4.1.1.83 #define MSG_PEEK 0x2

Peek at the incoming message.

4.1.1.84 #define MSG_DONTROUTE 0x4

Send without using routing tables.

4.1.1.85 #define MSG_DONTWAIT 0x20

Send/receive is nonblocking.

4.1.1.86 #define MSG_ZEROCOPYSEND 0x1000

Send with zero-copy.

4.1.1.87 #define QAPI_NET_WAIT_FOREVER (0xFFFFFFFF)

Infinite time for the timeout_ms argument in qapi_select().

4.1.1.88 #define FD_ZERO(*set*) qapi_fd_zero((set))

Clears a set.

4.1.1.89 #define FD_CLR(*handle, set*) qapi_fd_clr((handle), (set))

Removes a given file descriptor from a set.

4.1.1.90 #define FD_SET(*handle, set*) qapi_fd_set((handle), (set))

Adds a given file descriptor from a set.

4.1.1.91 #define FD_ISSET(*handle, set*) qapi_fd_isset((handle), (set))

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

4.1.2 Data Structure Documentation

4.1.2.1 struct in_addr

IPv4 Internet address.

Data fields

Type	Parameter	Description
uint32_t	s_addr	IPv4 address in network order.

4.1.2.2 struct sockaddr_in

BSD-style socket IPv4 Internet address.

Data fields

Type	Parameter	Description
uint16_t	sin_family	AF_INET.
uint16_t	sin_port	UDP/TCP port number in network order.
struct in_addr	sin_addr	IPv4 address in network order.
uint8_t	sin_zero	Reserved – must be zero.

4.1.2.3 struct in6_addr

IPv6 Internet address.

Data fields

Type	Parameter	Description
uint8_t	s_addr	128-bit IPv6 address.

4.1.2.4 struct ip46addr_n

BSD-style socket IPv6 Internet address.

Data fields

Type	Parameter	Description
uint16_t	type	AF_INET or AF_INET6.
union ip46addr-_n	a	Address union.
union ip46addr-_n	g	Gateway union.
uint32_t	subnet	Subnet.

4.1.2.5 union ip46addr_n.a

Data fields

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

4.1.2.6 union ip46addr_n.g

Data fields

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

4.1.2.7 struct sockaddr_in6

Socket address information.

Data fields

Type	Parameter	Description
uint16_t	sin_family	AF_INET6.
uint16_t	sin_port	UDP/TCP port number in network order.
uint32_t	sin_flowinfo	IPv6 flow information.

Type	Parameter	Description
struct in6_addr	sin_addr	IPv6 address.
int32_t	sin_scope_id	Set of interfaces for a scope.

4.1.2.8 struct ip46addr

Socket IPv4/IPv6 Internet address union.

Data fields

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

4.1.2.9 union ip46addr.a

Data fields

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

4.1.2.10 struct sockaddr

Generic socket Internet address.

Data fields

Type	Parameter	Description
uint16_t	sa_family	Address family.
uint16_t	sa_port	Port number in network order.
uint8_t	sa_data	Big enough for 16-byte IPv6 address.

4.1.2.11 struct sockaddr_ep

Exclude list endpoint.

Data fields

Type	Parameter	Description
struct sockaddr-_ep *	sockaddr_ep_-next	Next endpoint.
struct sockaddr *	sockaddr_ep_-addr	Endpoint address in exclude list.

4.1.2.12 struct fd_set

File descriptor sets for qapi_select().

Data fields

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

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	*family*	Protocol family used for communication. The supported families are: • AF_INET – IPv4 Internet protocols • AF_INET6 – IPv6 Internet protocols
in	*type*	Transport mechanism used for communication. The supported types are: • SOCK_STREAM – TCP • SOCK_DGRAM – UDP
in	*protocol*	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	*handle*	Socket handle returned from qapi_socket().
in	*addr*	Pointer to an address to be assigned to the socket. The actual address structure passed for the addr argument will depend on the address family.
in	*addrlen*	Specifies the size, in bytes, of the address pointed to by addr.

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	*handle*	Handle (returned from qapi_socket()) that refers to a SOCK_STREAM socket.
in	*backlog*	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	*handle*	Socket handle that has been created with qapi_socket(), bound to a local address with qapi_bind(), and listens for connections after qapi_listen().
in	*cliaddr*	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	*addrlen*	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, *svraddr is the address to which datagrams are sent by default, and the only address from which datagrams are received. If the socket is of type SOCK_STREAM, this call attempts to make a connection to the socket that is bound to the address specified by *srvaddr.

Parameters

in	*handle*	Socket handle returned from qapi_socket().
in	*srvaddr*	Pointer to the peer's address to which the socket is connected.
in	*addrlen*	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	*handle*	Socket handle returned from qapi_socket().
in	*level*	Protocol level at which the option exists.
in	*optname*	Name of the option.
in	*optval*	Pointer to the option value to be set.
in	*optlen*	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	*handle*	Socket handle returned from qapi_socket().
in	*level*	Protocol level at which the option exists.
in	*optname*	Name of the option.
in	*optval*	Pointer to a buffer in which the value for the requested option is to be returned.
in	*optlen*	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	*handle*	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	*handle*	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 (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	*handle*	Socket handle returned from qapi_socket().
in	*buf*	Pointer to a buffer for the received message.
in	*len*	Number of bytes to receive.
in	*flags*	0, or it is formed by ORing one or more of: • MSG_PEEK – Causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data. • MSG_OOB – Requests receipt of out-of-band data that would not be received in the normal data stream. • MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.
in	*from*	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	*fromlen*	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	*handle*	Socket handle returned from qapi_socket().
in	*buf*	Pointer to a buffer for the received message.
in	*len*	Number of bytes to receive.
in	*flags*	0, or it is formed by ORing one or more of: • MSG_PEEK – Causes the receive operation to return data from the beginning of the receive queue without removing that data from the queue. Thus, a subsequent receive call will return the same data. • MSG_OOB – Requests receipt of out-of-band data that would not be received in the normal data stream. • MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK.

Returns

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

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	*handle*	Socket handle returned from qapi_socket().
in	*buf*	Pointer to a buffer containing the message to be sent.
in	*len*	Number of bytes to send.
in	*flags*	0, or it is formed by ORing one or more of: • MSG_OOB – Sends out-of-band data on sockets that support this notion (e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data. • MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK. • MSG_DONTROUTE – Don not use a gateway to send the packet; only send it to hosts on directly-connected networks. This is usually used only by diagnostic or routing programs.
in	*to*	Pointer to the address of the target.
in	*tolen*	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	*handle*	Socket handle returned from qapi_socket().
in	*buf*	Pointer to a buffer containing message to be sent.
in	*len*	Number of bytes to send.
in	*flags*	0, or it is formed by ORing one or more of: • MSG_OOB – Sends out-of-band data on sockets that support this notion (e.g., of type SOCK_STREAM); the underlying protocol must also support out-of-band data. • MSG_DONTWAIT – Enables a nonblocking operation; if the operation blocks, the call fails with the error EAGAIN or EWOULDBLOCK. • MSG_DONTROUTE – Do not use a gateway to send the packet; only send it to hosts on directly-connected networks. This is usually used only by diagnostic or routing programs.

Returns

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

Parameters

in	*rd*	Pointer to a list of read socket handles.
in	*wr*	Pointer to a list of write socket handles.
in	*ex*	Pointer to a list of exception socket handles.
in	*timeout_ms*	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	*set*	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	*handle*	Socket handle returned from qapi_socket().
in	*set*	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	*handle*	Socket handle returned from qapi_socket().
in	*set*	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	*handle*	Socket handle returned from qapi_socket().
in	*set*	Pointer to a list of sockets.

Returns

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

4.20 Get the Address of a Connected Peer

4.20.1 Function Documentation

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

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

Parameters

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

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). See Appendix A for TLS/DTLS supported ciphersuites.

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

This chapter contains the following sections:

5.1 QAPI SSL Data Types

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

5.1.1 Define Documentation

5.1.1.1 #define QAPI_NET_SSL_MAX_CERT_NAME_LEN (64)

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

5.1.1.2 #define QAPI_NET_SSL_MAX_DOMAIN_NAME_LEN (64)

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

5.1.1.3 #define QAPI_NET_SSL_MAX_NUM_CERTS (10)

Maximum number of file names returned in the qapi_Net_SSL_Cert_List() API.

5.1.1.4 #define QAPI_NET_SSL_CIPHERSUITE_LIST_DEPTH 8

Maximum number of cipher suites that can be configured.

5.1.1.5 #define QAPI_NET_SSL_INVALID_HANDLE (0)

Invalid handle.

5.1.1.6 #define QAPI_NET_SSL_PROTOCOL_UNKNOWN 0x00

Unknown SSL protocol version.

5.1.1.7 #define QAPI_NET_SSL_PROTOCOL_TLS_1_0 0x31

TLS version 1.0.

5.1.1.8 #define QAPI_NET_SSL_PROTOCOL_TLS_1_1 0x32

TLS version 1.1.

5.1.1.9 #define QAPI_NET_SSL_PROTOCOL_TLS_1_2 0x33

TLS version 1.2.

5.1.1.10 #define QAPI_NET_SSL_PROTOCOL_DTLS_1_0 0xEF

DTLS version 1.0.

5.1.1.11 #define QAPI_NET_SSL_PROTOCOL_DTLS_1_2 0xED

DTLS version 1.2.

5.1.1.12 #define QAPI_NET_TLS_PSK_WITH_RC4_128_SHA 0x008A

TLS PSK with RC4 128 SHA.

5.1.1.13 #define QAPI_NET_TLS_PSK_WITH_3DES_EDE_CBC_SHA 0x008B

TLS PSK with 3DES EDE CBC SHA.

5.1.1.14 #define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA 0x008C

TLS PSK with AES 128 CBC SHA.

5.1.1.15 #define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA 0x008D

TLS PSK with AES 256 CBC SHA.

5.1.1.16 #define QAPI_NET_TLS_PSK_WITH_AES_128_GCM_SHA256 0x00A8

TLS PSK with AES_128 GCM SHA256.

5.1.1.17 #define QAPI_NET_TLS_PSK_WITH_AES_256_GCM_SHA384 0x00A9

TLS PSK with AES 256 GCM SHA384.

5.1.1.18 #define QAPI_NET_TLS_PSK_WITH_AES_128_CBC_SHA256 0x00AE

TLS PSK with AES 128 CBC SHA256.

5.1.1.19 #define QAPI_NET_TLS_PSK_WITH_AES_256_CBC_SHA384 0x00AF

TLS PSK with AES 256 CBC SHA384.

5.1.1.20 #define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA 0x002F

Cipher TLS RSA with AES 128 CBC SHA.

5.1.1.21 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA 0x0033

Cipher TLS DHE RSA with AES 128 CBC SHA.

5.1.1.22 #define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA 0x0035

Cipher TLS RSA with AES 256 CBC SHA.

5.1.1.23 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA 0x0039

Cipher TLS DHE RSA with AES 256 CBC SHA.

5.1.1.24 #define QAPI_NET_TLS_RSA_WITH_AES_128_CBC_SHA256 0x003C

Cipher TLS RSA with AES 128 CBC SHA256.

5.1.1.25 #define QAPI_NET_TLS_RSA_WITH_AES_256_CBC_SHA256 0x003D

Cipher TLS RSA with AES 256 CBC SHA256.

5.1.1.26 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CBC_SHA256 0x0067

Cipher TLS DHE RSA with AES 128 CBC SHA256.

5.1.1.27 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 0x006B

Cipher TLS DHE RSA with AES 256 CBC SHA256.

5.1.1.28 #define QAPI_NET_TLS_RSA_WITH_AES_128_GCM_SHA256 0x009C

Cipher TLS RSA with AES 128 GCM SHA256.

5.1.1.29 #define QAPI_NET_TLS_RSA_WITH_AES_256_GCM_SHA384 0x009D

Cipher TLS RSA with AES 256 GCM SHA384.

5.1.1.30 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 0x009E

Cipher TLS DHE RSA with AES 128 GCM SHA256.

5.1.1.31 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 0x009F

Cipher TLS DHE RSA with AES 256 GCM SHA384.

5.1.1.32 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA 0xC004

Cipher TLS ECDH ECDSA with AES 128 CBC SHA.

5.1.1.33 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA 0xC005

Cipher TLS ECDH ECDSA with AES 256 CBC SHA.

5.1.1.34 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA 0xC009

Cipher TLS ECDHE ECDSA with AES 128 CBC SHA.

5.1.1.35 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA 0xC00A

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA.

5.1.1.36 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA 0xC00E

Cipher TLS ECDH RSA with AES 128 CBC SHA.

5.1.1.37 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA 0xC00F

Cipher TLS ECDH RSA with AES 256 CBC SHA.

5.1.1.38 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA 0xC013

Cipher TLS ECDHE RSA with AES 128 CBC SHA.

5.1.1.39 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA 0xC014

Cipher TLS ECDHE RSA with AES 256 CBC SHA.

5.1.1.40 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 0xC023

Cipher TLS ECDHE ECDSA with AES 128 CBC SHA256.

5.1.1.41 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384 0xC024

Cipher TLS ECDHE ECDSA with AES 256 CBC SHA384.

5.1.1.42 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256 0xC025

Cipher TLS ECDH ECDSA with AES 128 CBC SHA256.

5.1.1.43 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384 0xC026

Cipher TLS ECDH ECDSA with AES 256 CBC SHA384.

5.1.1.44 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 0xC027

Cipher TLS ECDHE RSA with AES 128 CBC SHA256.

5.1.1.45 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 0xC028

Cipher TLS ECDHE RSA with AES 256 CBC SHA384.

5.1.1.46 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256 0xC029

Cipher TLS ECDH RSA with AES 128 CBC SHA256.

5.1.1.47 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384 0xC02A

Cipher TLS ECDH RSA with AES 256 CBC SHA384.

5.1.1.48 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 0x-C02B

Cipher TLS ECDHE ECDSA with AES 128 GCM SHA256.

5.1.1.49 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 0x-C02C

Cipher TLS ECDHE ECDSA with AES 256 GCM SHA384.

5.1.1.50 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256 0x-C02D

Cipher TLS ECDH ECDSA with AES 128 GCM SHA256.

5.1.1.51 #define QAPI_NET_TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384 0x-C02E

Cipher TLS ECDH ECDSA with AES 256 GCM SHA384.

5.1.1.52 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 0xC02-F

Cipher TLS ECDHE RSA with AES 128 GCM SHA256.

5.1.1.53 #define QAPI_NET_TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 0xC030

Cipher TLS ECDHE RSA with AES 256 GCM SHA384.

5.1.1.54 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256 0xC031

Cipher TLS ECDH RSA with AES 128 GCM SHA256.

5.1.1.55 #define QAPI_NET_TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384 0xC032

Cipher TLS ECDH RSA with AES 256 GCM SHA384.

5.1.1.56 #define QAPI_NET_TLS_RSA_WITH_AES_128_CCM 0xC09C

Cipher TLS RSA with AES 128 CCM.

5.1.1.57 #define QAPI_NET_TLS_RSA_WITH_AES_256_CCM 0xC09D

Cipher TLS RSA with AES 256 CCM.

5.1.1.58 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM 0xC09E

Cipher TLS DHE RSA with AES 128 CCM.

5.1.1.59 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM 0xC09F

Cipher TLS DHE RSA with AES 256 CCM.

5.1.1.60 #define QAPI_NET_TLS_RSA_WITH_AES_128_CCM_8 0xC0A0

Cipher TLS RSA with AES 128 CCM 8.

5.1.1.61 #define QAPI_NET_TLS_RSA_WITH_AES_256_CCM_8 0xC0A1

Cipher TLS RSA with AES 256 CCM 8.

5.1.1.62 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_128_CCM_8 0xC0A2

Cipher TLS DHE RSA with AES 128 CCM 8.

5.1.1.63 #define QAPI_NET_TLS_DHE_RSA_WITH_AES_256_CCM_8 0xC0A3

Cipher TLS DHE RSA with AES 256 CCM 8.

5.1.1.64 #define QAPI_NET_TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC13

Cipher TLS ECDHE RSA with CHACHA20 POLY1305 SHA256.

5.1.1.65 #define QAPI_NET_TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 0xCC14

Cipher TLS ECDHE ECDSA with CHACHA20 POLY1305 SHA256.

5.1.1.66 #define QAPI_NET_TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 0xCC15

Cipher TLS DHE RSA with CHACHA20 POLY1305 SHA256.

5.1.1.67 #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_t

Structure to specify the certificate verification policy.

Data fields

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

5.1.2.2 struct qapi_Net_SSL_Identifier_t

Structure to get the identifier from the certificate.

Data fields

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

5.1.2.3 struct qapi_Net_SSL_Config_t

Structure to configure an SSL connection.

Data fields

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

5.1.2.4 struct qapi_Net_SSL_Cert_List_t

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

Data fields

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

5.1.2.5 struct qapi_Net_SSL_CERT_t

SSL client certificate info for conversion and storage.

Data fields

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

5.1.2.6 struct qapi_NET_SSL_CA_Info_t

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.

5.1.2.7 struct qapi_Net_SSL_CA_List_t

SSL certificate authority information for conversion and storage.

Data fields

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

5.1.2.8 struct qapi_Net_SSL_PSK_Table_t

SSL PSK table information for conversion and storage.

Data fields

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

5.1.2.9 struct qapi_Net_SSL_DI_Cert_t

SSL domain-issued certificate information for conversion and storage.

Data fields

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

5.1.2.10 struct __qapi_Net_SSL_Cert_Info_s

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

Data fields

Type	Parameter	Description
qapi_Net_SSL-_Cert_Type_t	cert_Type	Certification type.
union __qapi_-Net_SSL_Cert-_Info_s	info	Certificate information.

5.1.2.11 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.
qapi_Net_SSL-_DI_Cert_t	di_cert	Domain-issued certificate.

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. Not supported.
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.
QAPI_NET_SSL_DI_CERT_E Domain-issued certificate 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 foramt 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_DICERT_t

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

5.2.1.5 typedef const void* qapi_Net_SSL_CAList_t

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

Usage

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

5.2.1.6 typedef const void* qapi_Net_SSL_PSKTable_t

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

Identity_1: psk_key1

Identity_2: psk_key2

Usage

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

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 (client).

Parameters

in	*role*	Client role. Server is not supported.

Returns

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

Dependencies

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

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	*hdl*	SSL object handle.
in	*prot*	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	*ssl*	Connection handle.

in	*cfg*	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 or CA list, or a PSK table from nonvolatile memory.

Parameters

in	*name*	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	*type*	Type of data (certificate or CA list) to store. Could be either QAPI_NET_SSL_CERTIFICATE_E, QAPI_NET_SSL_CA_LIST_E, or QAPI_NET_SSL_PSK_TABLE_E.

Returns

0 on success.
Negative value on error (see SslErrors).

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 or CA list, or a PSK table in nonvolatile memory in encrypted form.

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

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

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

Identity_1: psk_key1

Identity_2: psk_key2

Parameters

in	*name*	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	*type*	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	*cert*	Address of the file containing the certificate in SSL internal format (*.bin file).
in	*size*	Size of the certificate file.

Returns

0 on success.
Negative value on error (see SslErrors).

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

Returns

0 on success.
Negative value on error (see SslErrors).

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

- Certficate – Loads a client certificate to the SSL object.
- Certificate Authority (CA) list – Enables the SSL object to perform certificate validation on the peer's certificate. Only one CA list can be set, thus the CA list must include all root certificates required for the Session
- PSK table – Holds a list of preshared keys (PSK) to load SSL conext. Only one PSK table can be set, thus the PSK table must include all PSK entries required for the session.

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

Parameters

in	*hdl*	SSL object handle.
in	*type*	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	*name*	Name of the file to load.

Returns

0 on success.
Negative value on error (see SslErrors).

5.10 Load an SSL Certificate and Return the Identifier

5.10.1 Function Documentation

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

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

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

Parameters

in	*hdl*	SSL object handle.
out	*identifier*	Type of certificate identifier.
in	*name*	Name of the file to load.

Returns

0 on success,
Negative value on error (see SslErrors).

5.11 Get a List of SSL Certificates

5.11.1 Function Documentation

5.11.1.1 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 or CA lists, or a PSK table stored in nonvolatile memory.

The structure qapi_Net_SSL_Cert_List_t must be allocated by the caller.

Parameters

in	*type*	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	*list*	List of file names.

Returns

Number of files.
0 on error.

5.12 Attach a Socket Descriptor to the SSL Connection

5.12.1 Function Documentation

5.12.1.1 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	*ssl*	SSL connection handle.
in	*fd*	FD socket descriptor.

Returns

QAPI_OK on success.
QAPI_ERR_INVALID_PARAM_SSL on error.

5.13 Accept an SSL Connection From the Client

5.13.1 Function Documentation

5.13.1.1 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	*ssl*	SSL connection handle.

Returns

QAPI_SSL_OK_HS on success.
QAPI_ERR_∗ on error.

5.14 Initiate an SSL Handshake

5.14.1 Function Documentation

5.14.1.1 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	*ssl*	SSL connection handle.

Returns

QAPI_SSL_OK_HS on success.
QAPI_ERR_∗ on error.

5.15 Close an SSL Connection

5.15.1 Function Documentation

5.15.1.1 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 socket QAPIs.

Parameters

in	*ssl*	SSL connection handle.

Returns

QAPI_OK on success.
QAPI_ERR_INVALID_PARAM_SSL on error (invalid connection handle).

5.16 Free an SSL Object Handle

5.16.1 Function Documentation

5.16.1.1 qapi_Status_t qapi_Net_SSL_Obj_Free (qapi_Net_SSL_Obj_Hdl_t *hdl*)

Frees the SSL object handle.

Parameters

in	*hdl*	SSL object handle.

Returns

QAPI_OK on success.

Dependencies

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

5.17 Read SSL Data

5.17.1 Function Documentation

5.17.1.1 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	*hdl*	Connection handle.
in,out	*buf*	Buffer to hold received data. Must be allocated by the application.
in	*size*	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.

5.18 Write SSL Data

5.18.1 Function Documentation

5.18.1.1 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	*hdl*	Connection handle.
in	*buf*	Buffer with the data to be sent.
in	*size*	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.

5.19 Determine Whether a Certificate File Exists

5.19.1 Function Documentation

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

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

Parameters

in	*file_name*	Certificate file name.
in	*type*	Type of SSL certificate.

Returns

QAPI_OK on success.
QAPI_ERR_* on error.

6 Qualcomm Secure/Trusted Execution Environment APIs

This chapter describes the Qualcomm secure execution environment (QSEE) and Qualcomm trusted execution environment (QTEE) APIs, and the IOpener API.

- Create an IOpener object
- Library Functions for Accessing the QSEECom Drivers
- IOpener Functions

6.1 Create an IOpener object

6.1.1 Function Documentation

6.1.1.1 int qapi_QTEEInvoke_GetOpener (Object * *ClientOpenerObj*)

Function used by clients to create an IOpener object. The call process is as follows:

```
{.c}
#include <object.h>
#include <stdint.h>

Object openerObj = Object_NULL;
Object myTAObj = Object_NULL;
int32_t err = Object_OK;
int32_t out = 0;

// Start the TA you desire
struct QSEECom_handle *clnt_handle = NULL;
err = qapi_QSEECom_start_app(&clnt_handle, "/firmware/image", "my_ta", 0);

// Get the opener object to open TA services with
err = qapi_QTEEInvoke_GetOpener(&openerObj);

// Open an object referencing the TA service of choice
err = qapi_IOpener_open(openerObj, CMyTA_UID, &myTAObj);

// Call TA method using RPC
err = IMyTA_add(myTAObj, 5, 10, &out);

// Release
IMyTA_release(myTAObj);
qapi_IOpener_release(openerObj);

// Unload TA
err = qapi_QSEECom_shutdown_app(&clnt_handle);
```


Parameters

out	*ClientOpenerObj*	Object to be used with IOpener.

Returns

0 on success; -1 on failure.

6.2 Library Functions for Accessing the QSEECom Drivers

6.2.1 Data Structure Documentation

6.2.1.1 struct qapi_QSEECom_handle_t

Handle to the loaded trusted application (TA).

This handle is returned by qapi_QSEECom_start_app().

Data fields

Type	Parameter	Description
unsigned char *	mem_sbuffer	Not used – set to NULL.

6.2.2 Function Documentation

6.2.2.1 int qapi_QSEECom_start_app (qapi_QSEECom_handle_t ** *clnt_handle*, const char * *path*, const char * *fname*, uint32_t *sb_size*)

Starts a trusted application.

Loads and starts a trusted application. The application is verified as secure by digital signature verification.

A trusted application is built using the QTEE tool chain. The binaries are split into multiple files and are saved to the file system.

Example:

The directory "/firmware/image" contains a trusted application named "my_ta". The trusted application is split into "my_ta.b00", "my_ta.b01", "my_ta.b02", "my_ta.b03", "my_ta.b04", "my_ta.b05", and "my_ta.mdt".

```
qapi_QSEECom_handle_t *handle = NULL;
qapi_QSEECom_start_app(&handle, "/firmware/image", "my_ta", 0);
...
qapi_QSEECom_shutdown_app(&handle);
```

Parameters

in,out	*clnt_handle*	Handle to the loaded trusted application.
in	*path*	Name of the directory that contains the trusted application.
in	*fname*	Name of the trusted application file name without the extension.
in	*sb_size*	Not used – set to 0.

Returns

Zero on success, negative value on failure.

6.2.2.2 int qapi_QSEECom_shutdown_app (qapi_QSEECom_handle_t ** *handle*)

Shuts down a started trusted application.

See qapi_QSEECom_start_app() for a usage example.

Parameters

in	*handle*	Handle to the loaded trusted application.

Returns

Zero on success, negative on failure.

6.3 IOpener Functions

6.3.1 Define Documentation

6.3.1.1 #define qapi_IOpener_ERROR_NOT_FOUND INT32_C(10)

Error code returned by qapi_IOpener_open() if a TA service with a UID specified by the id_val parameter is not found.

6.3.1.2 #define qapi_IOpener_ERROR_PRIVILEGE INT32_C(11)

Error code returned by qapi_IOpener_open() if the TA service's UID specified by the id_val parameter is other than one of the reserved IDs.

6.3.2 Function Documentation

6.3.2.1 static int32_t qapi_IOpener_release (Object *self*)

Releases an object in the ARM® TrustZone® (TZ) that was opened via qapi_IOpener_open().

Parameters

in	*self*	Object to be released.

Returns

0: success, -1: failure.

See also

IOpener_open()

6.3.2.2 static int32_t qapi_IOpener_retain (Object *self*)

Retains and increments the references on an object.

This is useful if the object is sent to a different thread or process. In that case, the sender can call retain before sending the object.

Parameters

in	*self*	Object to be retained or reference to be incremented.

Returns

0: success, -1: failure.

6.3.2.3 static int32_t qapi_IOpener_open (Object *self,* uint32_t *id_val,* Object * *obj_ptr*)

Opens a TZ object to be used for execution.

Allows the user to open a specific class ID or a TA service ID to then be used as desired in asking the TZ to execute on the user's behalf.

Parameters

in	*self*	Object to use for opening. This should be retrieved via qapi_QTEEInvoke_GetOpener().
in	*id_val*	ID of the TA service that is to be opened.
out	*obj_ptr*	Return object representing the requested TA service.

Returns

0: success, -1: failure.

7 QAPI Networking Services

This chapter describes the Networking Services and utilities QAPIs.

7.1 Networking Services Macros, Data Types, and Enumerations

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

7.1.1 Define Documentation

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

Verifies whether the IPv4 address is multicast.

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

Parameters

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

Returns

1 if the IPv4 address is multicast, 0 otherwise.

7.1.1.2 #define IF_NAMELEN 20

Default maximum length for interface names.

7.1.1.3 #define QAPI_NET_IPV4_MAX_ROUTES (3)

Maximum IPv4 routing configurations.

7.1.1.4 #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 1111111010, followed by 54 zeros, followed by 64 bits of the interface identifier.

Parameters

in	*ipv6_Address*	IPv6 address to check.

Returns

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

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

Checks whether the IPv6 address is multicast.

Parameters

in	*ipv6_Address*	IPv6 address to check.

Returns

1 if the IPv6 address is multicast, 0 otherwise.

7.1.1.6 #define QAPI_NET_IPV6_MAX_ROUTES (3)

Maximum IPv6 routing configurations.

7.1.1.7 #define QAPI_NET_IFNAME_LEN 12

Maximum length for the interface name.

7.1.2 Data Structure Documentation

7.1.2.1 struct qapi_Net_Ping_V4_t

IPv4 ping input.

Data fields

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

7.1.2.2 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.

7.1.2.3 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.

7.1.2.4 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.

7.1.2.5 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.

7.1.2.6 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.

7.1.2.7 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).

7.1.2.8 struct qapi_Ping_Info_Resp_s

Ping response structure.

Data fields

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

7.1.3 Enumeration Type Documentation

7.1.3.1 enum qapi_Net_Route_Command_t

Commands for routing QAPI net services.

Enumerator:

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

7.1.3.2 enum qapi_Net_IPv4cfg_Command_t

Commands for the IPv4 configuration QAPI.

Enumerator:

QAPI_NET_IPV4CFG_QUERY_E Get the IPv4 parameters of an interface, such as IP address, subnet mask, and default gateway.

QAPI_NET_IPV4CFG_STATIC_IP_E Assign the IPv4 address, subnet mask, and default gateway.

QAPI_NET_IPV4CFG_DHCP_IP_E Run the DHCPv4 client to obtain IPv4 parameters from the DHCPv4 server.

QAPI_NET_IPV4CFG_AUTO_IP_E Run auto-IP (automatic private IP addressing).

7.2 Get the Names of All Network Interfaces

7.2.1 Function Documentation

7.2.1.1 int32_t qapi_Net_Get_All_Ifnames (qapi_Net_Ifnameindex_t * *if_Name_Index*)

Retrieves the textual names of all network interfaces.

Parameters

out	*if_Name_Index*	Array to contain the retrieved information.

Returns

Number of network interfaces

7.3 Parse an Address String into an IPv4/IPv6 Address

7.3.1 Function Documentation

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

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

Parameters

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

Returns

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

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

7.4.1 Function Documentation

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

Returns

Pointer to the resulting string if OK, otherwise NULL.

7.5 Get the Physical Address and Length of an Interface

7.5.1 Function Documentation

7.5.1.1 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	*interface_Name*	Name of the interface for which to retrive the physical address and or physical address length.
out	*address*	Pointer to where to save the address of the buffer containing the physical address.
out	*address_Len*	Pointer to where to store the physical address length.

Returns

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

7.6 Check Whether an Interface Exists

7.6.1 Function Documentation

7.6.1.1 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	*interface_Name*	Name of the interface for which to check whether it exists.

Returns

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

7.7 IPv4 Network Configuration

7.7.1 Function Documentation

7.7.1.1 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	*interface_Name*	Pointer to the interface name.
in	*cmd*	Command mode. Possible values are: • QAPI_NET_IPv4CFG_QUERY_E (0) – Get the IPv4 parameters of an interface. • QAPI_NET_IPv4CFG_STATIC_IP_E (1) – Assign the IPv4 address, subnet mask, and default gateway.
in	*ipv4_Addr*	Pointer to the IPv4 address in host order.
in	*subnet_Mask*	Pointer to the IPv4 subnet mask in host order.
in	*gateway*	Pointer to the IPv4 gateway address in host order.

Returns

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

7.8 Send an IPv4 Ping

7.8.1 Function Documentation

7.8.1.1 qapi_Status_t qapi_Net_Ping (uint32_t *ipv4_Addr,* uint32_t *size*)

Sends an IPv4 ping.

Parameters

in	*ipv4_Addr*	IPv4 destination address in network order.
in	*size*	Size of the ping payload in bytes.

Returns

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

7.9 Send an IPv4 Ping with a Response

7.9.1 Function Documentation

7.9.1.1 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	*ping_buf*	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	*ping_resp*	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.

7.10 IPv4 Route Commands

7.10.1 Function Documentation

7.10.1.1 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	*interface_Name*	Pointer to the interface name.
in	*cmd*	Command mode. Possible values are: • QAPI_NET_ROUTE_ADD_E (0) – Add route. • QAPI_NET_ROUTE_DEL_E (1) – Delete route. • QAPI_NET_ROUTE_SHOW_E (2) – Show route.
in	*ipv4_Addr*	Pointer to the IPv4 address in host order.
in	*subnet_Mask*	Pointer to the IPv4 subnet mask in host order.
in	*gateway*	Pointer to the IPv4 gateway address in host order.
in	*route_List*	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.

7.11 Send an IPv6 Ping

7.11.1 Function Documentation

7.11.1.1 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	*ipv6_Addr*	IPv6 address to which to send a ping.
in	*size*	Number of data bytes to send.
in	*interface_Name*	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.

7.12 Send an IPv6 Ping with a Response

7.12.1 Function Documentation

7.12.1.1 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	*ping6_buf*	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	*ping_resp*	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.

7.13 Get the IPv6 Address of an Interface

7.13.1 Function Documentation

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

Returns

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

7.14 IPv6 Route Commands

7.14.1 Function Documentation

7.14.1.1 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	*interface_Name*	Pointer to the name of the network interface.
in	*cmd*	Command mode. Possible values are: • QAPI_NET_ROUTE_ADD_E (0) – Add route • QAPI_NET_ROUTE_DEL_E (1) – Delete route • QAPI_NET_ROUTE_SHOW_E (2) – Show route
in	*ipv6_Addr*	Pointer to the IPv6 address.
in	*prefix_Length*	Pointer to the IPv6 prefix length.
in	*next_Hop*	Pointer to the IPv6 gateway address.
in	*route_List*	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.

7.15 Get the Interface Scope ID

7.15.1 Function Documentation

7.15.1.1 qapi_Status_t qapi_Net_IPv6_Get_Scope_ID (const char * *interface_Name,* int32_t * *scope_ID*)

Returns the scope ID for the interface.

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

Parameters

in	*interface_Name*	Pointer to the name of the interface for which to retrieve the scope ID.
out	*scope_ID*	Pointer to the location store the scope ID.

Returns

0 on success, or a negative error code.

8 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

8.1 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.

8.1.1 Define Documentation

8.1.1.1 #define MAX_DNS_SVR_NUM 4

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

8.1.1.2 #define QAPI_DNS_PORT 53

DNS server port.

8.1.1.3 #define QAPI_NET_DNS_V4_PRIMARY_SERVER_ID 0

DNS IPv4 primary server ID.

8.1.1.4 #define QAPI_NET_DNS_V4_SECONDARY_SERVER_ID 1

DNS IPv4 secondary server ID.

8.1.1.5 #define QAPI_NET_DNS_V6_PRIMARY_SERVER_ID 2

DNS IPv6 primary server ID.

8.1.1.6 #define QAPI_NET_DNS_V6_SECONDARY_SERVER_ID 3

DNS IPv6 secondary server ID.

8.1.1.7 #define gethostbyname(__*name*) qapi_Net_DNSc_Get_Host_By_Name(__name)

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

8.1.2 Data Structure Documentation

8.1.2.1 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.

8.1.2.2 struct qapi_hostent_s

Data structure returned from qapi_gethostbyname() or qapi_gethostbyname2(). Same as the UNIX struct hostent{}.

Data fields

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

8.1.3 Enumeration Type Documentation

8.1.3.1 enum qapi_Net_DNS_Command_t

Commands to start/stop/disable a DNS client.

Enumerator:

QAPI_NET_DNS_DISABLE_E Functionality is deprecated. Do not use.

QAPI_NET_DNS_START_E Allocate space for internal data structures when called for the first time. For subsequent calls, increases the ref count; DNS query is allowed after the start command. Processes DNS responses from the server.

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

8.2 Check Whether the DNS Client has Started

8.2.1 Function Documentation

8.2.1.1 int32_t qapi_Net_DNSc_Is_Started (void)

Checks whether the DNS client has started.

Returns

0 if not started or 1 if started.

8.3 Start, Stop, or Disable the DNS Client

8.3.1 Function Documentation

8.3.1.1 int32_t qapi_Net_DNSc_Command (qapi_Net_DNS_Command_t *cmd*)

Starts, stops, or disables the DNS client.

Parameters

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

Returns

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

8.4 Convert an IP Address Text String into an IP Address

8.4.1 Function Documentation

8.4.1.1 int32_t qapi_Net_DNSc_Reshost (char * *hostname,* struct ip46addr * *ipaddr*)

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

Parameters

in	*hostname*	Pointer to an IP address string or host name string.
in	*ipaddr*	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.

8.5 Convert an IP Address Text String for an Interface

8.5.1 Function Documentation

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

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

Parameters

in	*hostname*	Pointer to an IP address string or host name string.
in	*addr*	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	*iface_index*	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.

8.6 Get a List of DNS Servers

8.6.1 Function Documentation

8.6.1.1 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	*svr_list*	Pointer to a buffer to contain the list.
in	*iface_index*	Index of the configured DNS servers.

Returns

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

8.7 Get Index for Added DNS Server

8.7.1 Function Documentation

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

Returns

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

8.8 Add a DNS Server

8.8.1 Function Documentation

8.8.1.1 int32_t qapi_Net_DNSc_Add_Server (char * *svr_addr,* uint32_t *id*)

Adds a DNS server to the system.

Parameters

in	*svr_addr*	Pointer to the DNS server's IP address string.
in	*id*	Server ID can be QAPI_NET_DNS_V4_PRIMARY_SERVER_ID, QAPI_NET_DNS_V4_SECONDARY_SERVER_ID, QAPI_NET_DNS_V6_PRIMARY_SERVER_ID, QAPI_NET_DNS_V6_SECONDARY_SERVER_ID, or QAPI_NET_DNS_ANY_SERVER_ID.

Returns

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

8.9 Add a DNS Server to an Interface

8.9.1 Function Documentation

8.9.1.1 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	*svr_addr*	Pointer to DNS server's IP address string.
in	*id*	Server ID can be QAPI_NET_DNS_V4_PRIMARY_SERVER_ID, QAPI_NET_DNS_V4_SECONDARY_SERVER_ID, QAPI_NET_DNS_V6_PRIMARY_SERVER_ID, QAPI_NET_DNS_V6_SECONDARY_SERVER_ID, or QAPI_NET_DNS_ANY_SERVER_ID.
in	*iface*	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.

8.10 Remove a DNS Server

8.10.1 Function Documentation

8.10.1.1 int32_t qapi_Net_DNSc_Del_Server (uint32_t *id*)

Removes a DNS server from the system.

Parameters

in	*id*	Server ID, can be: QAPI_NET_DNS_V4_PRIMARY_SERVER_ID, QAPI_NET_DNS_V4_SECONDARY_SERVER_ID, QAPI_NET_DNS_V6_PRIMARY_SERVER_ID, QAPI_NET_DNS_V6_SECONDARY_SERVER_ID, or QAPI_NET_DNS_ANY_SERVER_ID

Returns

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

8.11 Removes a DNS Server from an Interface

8.11.1 Function Documentation

8.11.1.1 int32_t qapi_Net_DNSc_Del_Server_on_iface (uint32_t *id,* char * *iface_index*)

Removes a DNS server from an interface.

Parameters

in	*id*	Server ID, can be: QAPI_NET_DNS_V4_PRIMARY_SERVER_ID, QAPI_NET_DNS_V4_SECONDARY_SERVER_ID, QAPI_NET_DNS_V6_PRIMARY_SERVER_ID, QAPI_NET_DNS_V6_SECONDARY_SERVER_ID, or QAPI_NET_DNS_ANY_SERVER_ID
in	*iface_index*	Name of interface from which to delete a DNS server.

Returns

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

8.12 Get IPv4 Host Information by Name

8.12.1 Function Documentation

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

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	*name*	Pointer to either a host name or an IPv4 address in standard dot notation.
out	*ipaddr*	Resolved host information.

Returns

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

8.13 Get IPv4/IPv6 Host Information by Name

8.13.1 Function Documentation

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

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

Returns

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

9 MQTT API

This chapter describes the MQTT API.

- MQTT Data Types
- MQTT APIs

9.1 MQTT Data Types

Net MQTT Length Defines

- #define QAPI_NET_MQTT_MAX_CLIENT_ID_LEN 128
- #define QAPI_NET_MQTT_MAX_TOPIC_LEN 128

9.1.1 Define Documentation

9.1.1.1 #define QAPI_NET_MQTT_MAX_CLIENT_ID_LEN 128

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

9.1.1.2 #define QAPI_NET_MQTT_MAX_TOPIC_LEN 128

Maximum topic length.

9.1.1.3 #define qapi_Net_MQTT_Pass_Pool_Ptr(*a, b*) mqtt_set_byte_pool(a,b)

Macro that passes a Byte Pool pointer for the MQTT application.

Parameter a – Handle.

Parameter b – Pointer to the Byte Pool.

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

Note: This macro is only used in the DAM space.

9.1.1.4 #define qapi_Net_MQTT_Destroy(*a*) mqtt_destroy_indirection(a)

Macro that releases a Byte Pool pointer for the MQTT application.

Parameter a – Handle.

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

Note: This macro is only used in the DAM space.

9.1.2 Data Structure Documentation

9.1.2.1 struct qapi_Net_MQTT_config_s

MQTT configuration.

Data fields

Type	Parameter	Description
struct sockaddr	local	MQTT client IP address and port number.
struct sockaddr	remote	MQTT server IP address and port number.
bool	nonblocking_-connect	Blocking or nonblocking MQTT connection.
uint8_t	client_id	MQTT vlient ID.

Type	Parameter	Description
int32_t	client_id_len	MQTT client ID length.
uint32_t	keepalive_-duration	Conection keepalive duration in seconds.
uint8_t	clean_session	Clean session flag; 0 – No clean session, 1 – clean session.
uint8_t *	will_topic	Will topic name.
int32_t	will_topic_len	Will topic length.
uint8_t *	will_message	Will message.
int32_t	will_message_-len	Will message length.
uint8_t	will_retained	Will retain flag.
uint8_t	will_qos	Will QOS.
uint8_t *	username	Client username.
int32_t	username_len	Client user length.
uint8_t *	password	Client password.
int32_t	password_len	Client password length.
uint32_t	connack_timed-_out_sec	Timeout value for which the client waits for the CONNACK packet from the server.
qapi_Net_SSL-_Config_t	ssl_cfg	SSL configuration.
qapi_Net_SSL-_CAList_t	ca_list	SSL CA cert details.
qapi_Net_SSL-_Cert_t	cert	SSL cert details.

9.1.3 Enumeration Type Documentation

9.1.3.1 enum QAPI_NET_MQTT_SUBSCRIBE_CBK_MSG

Reason codes for a subscription callback.

Enumerator:

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

9.1.3.2 enum QAPI_NET_MQTT_CONNECT_CBK_MSG

Connection callback messages.

Enumerator:

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

9.1.3.3 enum QAPI_NET_MQTT_CONN_STATE

Connection states.

Enumerator:

QAPI_NET_MQTT_ST_DORMANT_E Connection is idle.
QAPI_NET_MQTT_ST_TCP_CONNECTING_E TCP is connecting.
QAPI_NET_MQTT_ST_TCP_CONNECTED_E TCP is connected.
QAPI_NET_MQTT_ST_SSL_CONNECTING_E SSL is connecting.
QAPI_NET_MQTT_ST_SSL_CONNECTED_E SSL is connected.
QAPI_NET_MQTT_ST_MQTT_CONNECTING_E MQTT is connecting.
QAPI_NET_MQTT_ST_MQTT_CONNECTED_E MQTT is connected.
QAPI_NET_MQTT_ST_MQTT_TERMINATING_E MQTT connection is terminating.
QAPI_NET_MQTT_ST_SSL_TERMINATING_E SSL connection is terminating.
QAPI_NET_MQTT_ST_TCP_TERMINATING_E TCP connection is terminating.
QAPI_NET_MQTT_ST_DYING_E MQTT connection is dying.
QAPI_NET_MQTT_ST_DEAD_E MQTT connection is dead.

9.1.3.4 enum QAPI_NET_MQTT_MSG_TYPES

MQTT message types.

Enumerator:

QAPI_NET_MQTT_CONNECT Connect.
QAPI_NET_MQTT_CONNACK Connection acknowledgement.
QAPI_NET_MQTT_PUBLISH Publish.
QAPI_NET_MQTT_PUBACK Publish acknowledgement.
QAPI_NET_MQTT_PUBREC PubRec.
QAPI_NET_MQTT_PUBREL PubRel.
QAPI_NET_MQTT_PUBCOMP PubComp.
QAPI_NET_MQTT_SUBSCRIBE Subscribe.
QAPI_NET_MQTT_SUBACK Subscribe acknowledgement.
QAPI_NET_MQTT_UNSUBSCRIBE Unsubscribe.
QAPI_NET_MQTT_UNSUBACK Unsubscribe acknowledgement.
QAPI_NET_MQTT_PINGREQ Ping request.
QAPI_NET_MQTT_PINGRESP Ping response.
QAPI_NET_MQTT_DISCONNECT Disconnect.
QAPI_NET_MQTT_MQTT_NO_RESPONSE_MSG_REQD No response message is required.
QAPI_NET_MQTT_INVALID_RESP Invalid response.

9.2 MQTT APIs

9.2.1 Function Documentation

9.2.1.1 qapi_Status_t qapi_Net_MQTT_New (qapi_Net_MQTT_Hndl_t * *hndl*)

Creates a new MQTT context.

Parameters

out	*hndl*	Newly created MQTT context.

Returns

QAPI_OK on success, QAPI_ERROR on failure.

9.2.1.2 qapi_Status_t qapi_Net_MQTT_Destroy (qapi_Net_MQTT_Hndl_t *hndl*)

Destroys an MQTT context.

Parameters

in	*hndl*	Handle for the MQTT context to be destroyed.

Returns

QAPI_OK on success or QAPI_ERROR on failure.

9.2.1.3 qapi_Status_t qapi_Net_MQTT_Connect (qapi_Net_MQTT_Hndl_t *hndl,* const qapi_Net_MQTT_Config_t * *config*)

Connects to an MQTT broker.

Parameters

in	*hndl*	MQTT handle.
in	*config*	MQTT client configuration.

Returns

QAPI_OK on success or < 0 on failure.

9.2.1.4 qapi_Status_t qapi_Net_MQTT_Disconnect (qapi_Net_MQTT_Hndl_t *hndl*)

Disconnects from an MQTT broker.

Parameters

in	*hndl*	MQTT handle.

Returns

QAPI_OK on success or < 0 on failure.

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

Publishes a message to a particular topic.

Parameters

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

Returns

QAPI_OK on success or <0 on failure.

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

Publishes a message to a particular topic.

Parameters

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

Returns

QAPI_OK on success or <0 on failure.

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

Subscribes to a topic.

Parameters

in	*hndl*	MQTT handle.
in	*topic*	Subscription topic.
in	*qos*	QOS to be used.

Returns

QAPI_OK on success or < 0 on failure.

9.2.1.8 qapi_Status_t qapi_Net_MQTT_Unsubscribe (qapi_Net_MQTT_Hndl_t *hndl,* const uint8_t * *topic*)

Unsubscribes from a topic.

Parameters

in	*hndl*	MQTT handle
in	*topic*	Topic from which to unsubscribe.

Returns

QAPI_OK on success or < 0 on failure.

9.2.1.9 qapi_Status_t qapi_Net_MQTT_Set_Connect_Callback (qapi_Net_MQTT_-Hndl_t *hndl,* qapi_Net_MQTT_Connect_CB_t *callback*)

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

Parameters

in	*hndl*	MQTT handle.
in	*callback*	Callback to be invoked.

Returns

Success or failure.

9.2.1.10 qapi_Status_t qapi_Net_MQTT_Set_Subscribe_Callback (qapi_Net_MQTT_Hndl_t *hndl,* qapi_Net_MQTT_Subscribe_CB_t *callback*)

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

Parameters

in	*hndl*	MQTT handle.
in	*callback*	Callback to be invoked.

Returns

QAPI_OK on success or < 0 on failure.

9.2.1.11 qapi_Status_t qapi_Net_MQTT_Set_Message_Callback (qapi_Net_MQTT_Hndl_t *hndl,* qapi_Net_MQTT_Message_CB_t *callback*)

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

Parameters

in	*hndl*	MQTT handle.
in	*callback*	Callback to be invoked.

Returns

QAPI_OK on success or < 0 on failure.

9.2.1.12 qapi_Status_t qapi_Net_MQTT_Set_Publish_Callback (qapi_Net_MQTT_Hndl_t *hndl,* qapi_Net_MQTT_Publish_CB_t *callback*)

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

Parameters

in	*hndl*	MQTT handle.
in	*callback*	Callback to be invoked.

Returns

QAPI_OK on success or < 0 on failure.

9.2.1.13 qapi_Status_t qapi_Net_MQTT_Allow_Unsub_Publish (qapi_Net_MQTT_Hndl_t *hndl,* bool *allow_unsub_pub*)

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

Parameters

in	*hndl*	MQTT handle.
in	*allow_unsub_pub*	condition that allows this behaviour.

Returns

QAPI_OK on success or < 0 on failure.

10 HTTP(S) APIs

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

- HTTP(S) API

10.1 HTTP(S) API

10.1.1 Define Documentation

10.1.1.1 #define qapi_Net_HTTPc_Pass_Pool_Ptr(*a, b*) httpc_set_byte_pool(a,b)

Macro that passes a Byte Pool pointer for the HTTP client.

Parameter a – Handle.

Parameter b – Pointer to the Byte Pool.

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

Note: This macro is only used in the DAM space.

10.1.1.2 #define qapi_Net_HTTPc_Free_sess(*a*) httpc_destroy_indirection(a, TXM_QAPI_HTTPC_FREE_SESSION)

Macro that releases a Byte Pool pointer for the HTTP client.

Parameter a – Handle.

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

Note: This macro is only used in the DAM space.

10.1.2 Data Structure Documentation

10.1.2.1 struct qapi_Net_HTTPc_Response_t

HTTP response data returned by qapi_HTTPc_CB_t().

Data fields

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

10.1.2.2 struct qapi_Net_HTTPc_Sock_Opts_t

HTTP socket options.

Data fields

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

10.1.2.3 struct qapi_Net_HTTPc_Config_t

Structure to configure an HTTP client session.

Data fields

Type	Parameter	Description
uint16_t	addr_type	Address type AF_UNSPEC, AF_INET or AF_INET6 (used for DNS resolution only).
uint32_t	sock_options_-cnt	Number of socket options.
qapi_Net_HTT-Pc_Sock_Opts-_t *	sock_options	Socket options – only the Linger option is currently supported.
uint16_t	max_send_-chunk	Maximum send data chunk per transaction.
uint16_t	max_send_-chunk_delay_-ms	Maximum delay between send data chunks (msec).
uint8_t	max_send_-chunk_retries	Maximum send data chunk retries.
uint8_t	max_conn_-poll_cnt	Maximum connect polling count.
uint32_t	max_conn_-poll_interval_-ms	Maximum connect polling interval

10.1.3 Typedef Documentation

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

HTTP response user callback registered during qapi_Net_HTTPc_New_sess().

Parameters

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

10.1.4 Enumeration Type Documentation

10.1.4.1 enum qapi_Net_HTTPc_Method_e

HTTP request types supported by qapi_Net_HTTPc_Request().

Enumerator:

QAPI_NET_HTTP_CLIENT_GET_E HTTP get request.
QAPI_NET_HTTP_CLIENT_POST_E HTTP post request.

QAPI_NET_HTTP_CLIENT_PUT_E HTTP put request.
QAPI_NET_HTTP_CLIENT_PATCH_E HTTP patch request.
QAPI_NET_HTTP_CLIENT_HEAD_E HTTP head request.
QAPI_NET_HTTP_CLIENT_CONNECT_E HTTP connect request.

10.1.4.2 enum qapi_Net_HTTPc_CB_State_e

HTTP callback state returned by qapi_HTTPc_CB_t().

Enumerator:

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

10.1.5 Function Documentation

10.1.5.1 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.

10.1.5.2 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.

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

Creates a new HTTP client session.

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

Parameters

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

Returns

On success, qapi_Net_HTTPc_handle_t is returned. NULL otherwise.

10.1.5.4 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	*handle*	Handle to the HTTP client session.

Returns

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

10.1.5.5 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	*handle*	Handle to the HTTP client session.
in	*URL*	Server URL informtion.
in	*port*	Server port information.

Returns

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

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

Connects an HTTP client session to the HTTP proxy server.

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

Parameters

in	*handle*	Handle to the HTTP client session.
in	*URL*	Server URL information.
in	*port*	Server port information.
in	*secure_proxy*	Secure proxy connection.

Returns

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

10.1.5.7 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	*handle*	Handle to the HTTP client session.

Returns

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

10.1.5.8 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.

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

Parameters

in	*handle*	Handle to the HTTP client session.
in	*cmd*	HTTP request method information.
in	*URL*	Server URL information.

Returns

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

10.1.5.9 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	*handle*	Handle to the HTTP client session.
in	*body*	HTTP body related information buffer.
in	*body_Length*	HTTP body buffer length.

Returns

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

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

Sets an HTTP client session parameter.

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

Parameters

in	*handle*	Handle to the HTTP client session.
in	*key*	HTTP key related information.
in	*value*	HTTP value associated with the key.

Returns

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

10.1.5.11 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	*handle*	Handle to the HTTP client session.
in	*type*	HTTP header type related information.
in	*value*	HTTP value associated with the header type.

Returns

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

10.1.5.12 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	*handle*	Handle to the HTTP client session.

Returns

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

10.1.5.13 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 session.

Invocation of this function configures the HTTP client SSL session.

Parameters

in	*handle*	Handle to the HTTP client session.
in	*ssl_Cfg*	SSL configuration information.

Returns

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

10.1.5.14 qapi_Status_t qapi_Net_HTTPc_Configure (qapi_Net_HTTPc_handle_t *handle,* qapi_Net_HTTPc_Config_t * *httpc_Cfg*)

Configures the HTTP client session based on the application requirement.

Parameters

in	*handle*	Handle to the HTTP client session.
in	*httpc_Cfg*	HTTP client configuration information.

Returns

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

11 QAPI Status and Error Codes

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

11.1 QAPI Status Codes

SSL Module Error Codes

- #define QAPI_ERR_SSL_CERT __QAPI_ERROR(QAPI_MOD_NETWORKING, 1)
- #define QAPI_ERR_SSL_CONN __QAPI_ERROR(QAPI_MOD_NETWORKING, 2)
- #define QAPI_ERR_SSL_HS_NOT_DONE __QAPI_ERROR(QAPI_MOD_NETWORKING, 3)
- #define QAPI_ERR_SSL_ALERT_RECV __QAPI_ERROR(QAPI_MOD_NETWORKING, 4)
- #define QAPI_ERR_SSL_ALERT_FATAL __QAPI_ERROR(QAPI_MOD_NETWORKING, 5)
- #define QAPI_SSL_HS_IN_PROGRESS __QAPI_ERROR(QAPI_MOD_NETWORKING, 6)
- #define QAPI_SSL_OK_HS __QAPI_ERROR(QAPI_MOD_NETWORKING, 7)
- #define QAPI_ERR_SSL_CERT_CN __QAPI_ERROR(QAPI_MOD_NETWORKING, 8)
- #define QAPI_ERR_SSL_CERT_TIME __QAPI_ERROR(QAPI_MOD_NETWORKING, 9)
- #define QAPI_ERR_SSL_CERT_NONE __QAPI_ERROR(QAPI_MOD_NETWORKING, 10)
- #define QAPI_ERR_SSL_NETBUF __QAPI_ERROR(QAPI_MOD_NETWORKING, 11)
- #define QAPI_ERR_SSL_SOCK __QAPI_ERROR(QAPI_MOD_NETWORKING, 12)

Generic Error Codes

- #define QAPI_NET_ERR_INVALID_IPADDR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,
  21)))
- #define QAPI_NET_ERR_CANNOT_GET_SCOPEID ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,
  22)))
- #define QAPI_NET_ERR_SOCKET_CMD_TIME_OUT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,
  23)))
- #define QAPI_NET_ERR_MAX_SERVER_REACHED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING,
  24)))

MQTT Error Codes

- #define QAPI_NET_MQTT_ERR_NUM_START 25
- #define QAPI_NET_MQTT_ERR_ALLOC_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START)
- #define QAPI_NET_MQTT_ERR_BAD_PARAM ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 1))
- #define QAPI_NET_MQTT_ERR_BAD_STATE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 2))
- #define QAPI_NET_MQTT_ERR_CONN_CLOSED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 3))
- #define QAPI_NET_MQTT_ERR_MSG_DESERIALIZATION_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 4))
- #define QAPI_NET_MQTT_ERR_MSG_SERIALIZATION_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 5))
- #define QAPI_NET_MQTT_ERR_NEGATIVE_CONNACK ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 6))
- #define QAPI_NET_MQTT_ERR_NO_DATA ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 7))
- #define QAPI_NET_MQTT_ERR_NONZERO_REFCOUNT ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 8))
- #define QAPI_NET_MQTT_ERR_PINGREQ_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 9))
- #define QAPI_NET_MQTT_ERR_PUBACK_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 10))
- #define QAPI_NET_MQTT_ERR_PUBCOMP_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 11))
- #define QAPI_NET_MQTT_ERR_PUBLISH_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 12))
- #define QAPI_NET_MQTT_ERR_PUBREC_MSG_CREATION_FAILED ((qapi_Status_t)__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_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 14))
- #define QAPI_NET_MQTT_ERR_RX_INCOMPLETE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 15))
- #define QAPI_NET_MQTT_ERR_SOCKET_FATAL_ERROR ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 16))
- #define QAPI_NET_MQTT_ERR_TCP_BIND_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 17))
- #define QAPI_NET_MQTT_ERR_TCP_CONNECT_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 18))
- #define QAPI_NET_MQTT_ERR_SSL_CONN_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 19))
- #define QAPI_NET_MQTT_ERR_SUBSCRIBE_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 21))
- #define QAPI_NET_MQTT_ERR_SUBSCRIBE_UNKNOWN_TOPIC ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 21))
- #define QAPI_NET_MQTT_ERR_UNSUBSCRIBE_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 22))
- #define QAPI_NET_MQTT_ERR_UNEXPECTED_MSG ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 23))
- #define QAPI_NET_MQTT_ERR_PARTIAL_SUBSCRIPTION_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 24))
- #define QAPI_NET_MQTT_ERR_RESTORE_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 25))
- #define QAPI_NET_MQTT_ERR_MAX_NUMS 26
- #define QAPI_NET_NIPD_FLOW_SUSPENDED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, 27))

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 add 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_GPIOINT** (9)
- #define **QAPI_MOD_BSP_PWM** (10)
- #define **QAPI_MOD_BSP_ERR** (11)
- #define **QAPI_MOD_BSP_DIAG** (12)
- #define **QAPI_MOD_BSP_OM_SMEM** (13)
- #define **QAPI_MOD_CRYPTO** (14)
- #define **QAPI_MOD_RIL** (18)
- #define **QAPI_MOD_BSP_PMIC** (21)

Common QAPI Status Codes

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

- #define QAPI_OK ((qapi_Status_t)(0))
- #define QAPI_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 1)))
- #define QAPI_ERR_INVALID_PARAM ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 2)))
- #define QAPI_ERR_NO_MEMORY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 3)))
- #define QAPI_ERR_NO_RESOURCE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 4)))
- #define QAPI_ERR_BUSY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 6)))
- #define QAPI_ERR_NO_ENTRY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 7)))
- #define QAPI_ERR_NOT_SUPPORTED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 8)))

- #define QAPI_ERR_TIMEOUT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 9)))
- #define QAPI_ERR_BOUNDS ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 10)))
- #define QAPI_ERR_BAD_PAYLOAD ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 11)))
- #define QAPI_ERR_EXISTS ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 12)))
- #define QAPI_ERR_NOT_INITIALIZED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 13)))
- #define QAPI_ERR_INVALID_STATE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 13)))
- #define QAPI_ERR_API_DEPRACATED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 14)))

11.1.1 Define Documentation

11.1.1.1 #define QAPI_ERR_SSL_CERT __QAPI_ERROR(QAPI_MOD_NETWORKING, 1)

Error in own certificate.

11.1.1.2 #define QAPI_ERR_SSL_CONN __QAPI_ERROR(QAPI_MOD_NETWORKING, 2)

Error with the SSL connection.

11.1.1.3 #define QAPI_ERR_SSL_HS_NOT_DONE __QAPI_ERROR(QAPI_MOD_NETWORKING, 3)

Handshake must be completed before the operation can be attempted.

11.1.1.4 #define QAPI_ERR_SSL_ALERT_RECV __QAPI_ERROR(QAPI_MOD_NETWORKING, 4)

Received an SSL warning alert message.

11.1.1.5 #define QAPI_ERR_SSL_ALERT_FATAL __QAPI_ERROR(QAPI_MOD_NETWORKING, 5)

Received an SSL fatal alert message.

11.1.1.6 #define QAPI_SSL_HS_IN_PROGRESS __QAPI_ERROR(QAPI_MOD_NETWORKING, 6)

Handshake is in progress.

11.1.1.7 #define QAPI_SSL_OK_HS __QAPI_ERROR(QAPI_MOD_NETWORKING, 7)

Handshake was successful.

11.1.1.8 #define QAPI_ERR_SSL_CERT_CN __QAPI_ERROR(QAPI_MOD_NETWORKING, 8)

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

11.1.1.9 #define QAPI_ERR_SSL_CERT_TIME __QAPI_ERROR(QAPI_MOD_NETWORKING, 9)

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

11.1.1.10 #define QAPI_ERR_SSL_CERT_NONE __QAPI_ERROR(QAPI_MOD_NETWORKING, 10)

The SSL certificate of the peer is not trusted.

11.1.1.11 #define QAPI_ERR_SSL_NETBUF __QAPI_ERROR(QAPI_MOD_NETWORKING, 11)

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

11.1.1.12 #define QAPI_ERR_SSL_SOCK __QAPI_ERROR(QAPI_MOD_NETWORKING, 12)

Socket error; use qapi_errno.h to check for the reason code.

11.1.1.13 #define QAPI_NET_ERR_INVALID_IPADDR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING, 21)))

IP address is invalid.

11.1.1.14 #define QAPI_NET_ERR_CANNOT_GET_SCOPEID ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING, 22)))

Failed to get the scope ID.

11.1.1.15 #define QAPI_NET_ERR_SOCKET_CMD_TIME_OUT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING, 23)))

Socket command timed out.

11.1.1.16 #define QAPI_NET_ERR_MAX_SERVER_REACHED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_NETWORKING, 24)))

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

11.1.1.17 #define QAPI_NET_MQTT_ERR_NUM_START 25

MQTT error number start.

11.1.1.18 #define QAPI_NET_MQTT_ERR_ALLOC_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START)

MQTT memory allocation failed.

11.1.1.19 #define QAPI_NET_MQTT_ERR_BAD_PARAM ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 1))

MQTT bad parameter while invoking the API.

11.1.1.20 #define QAPI_NET_MQTT_ERR_BAD_STATE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 2))

MQTT connection is in a bad state.

11.1.1.21 #define QAPI_NET_MQTT_ERR_CONN_CLOSED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 3))

MQTT connection is closed.

11.1.1.22 #define QAPI_NET_MQTT_ERR_MSG_DESERIALIZATION_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 4))

MQTT packet decode failed.

11.1.1.23 #define QAPI_NET_MQTT_ERR_MSG_SERIALIZATION_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 5))

MQTT packet encode failed.

11.1.1.24 #define QAPI_NET_MQTT_ERR_NEGATIVE_CONNACK ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 6))

MQTT negative CONNACK recevied.

11.1.1.25 #define QAPI_NET_MQTT_ERR_NO_DATA ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 7))

MQTT no data.

11.1.1.26 #define QAPI_NET_MQTT_ERR_NONZERO_REFCOUNT ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 8))

MQTT no zero reference count while disconnecting.

11.1.1.27 #define QAPI_NET_MQTT_ERR_PINGREQ_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 9))

MQTT ping request message creation failed.

11.1.1.28 #define QAPI_NET_MQTT_ERR_PUBACK_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 10))

MQTT PUBACK message creation failed.

11.1.1.29 #define QAPI_NET_MQTT_ERR_PUBCOMP_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 11))

MQTT PUBCOM message creation failed.

11.1.1.30 #define QAPI_NET_MQTT_ERR_PUBLISH_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 12))

MQTT publish message creation failed.

11.1.1.31 #define QAPI_NET_MQTT_ERR_PUBREC_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 13))

MQTT PUBREC message creation failed.

11.1.1.32 #define QAPI_NET_MQTT_ERR_PUBREL_MSG_CREATION_FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 14))

MQTT PUBREL message creation failed.

11.1.1.33 #define QAPI_NET_MQTT_ERR_RX_INCOMPLETE ((qapi_Status_t)__QAPI_-ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 15))

MQTT Rx is incomplete.

11.1.1.34 #define QAPI_NET_MQTT_ERR_SOCKET_FATAL_ERROR ((qapi_Status_t)_-_QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_-START + 16))

MQTT socket fatal error.

11.1.1.35 #define QAPI_NET_MQTT_ERR_TCP_BIND_FAILED ((qapi_Status_t)__QAPI-_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 17))

MQTT TCP bind error.

11.1.1.36 #define QAPI_NET_MQTT_ERR_TCP_CONNECT_FAILED ((qapi_Status_t)_-_QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_-START + 18))

MQTT TCP connection error.

11.1.1.37 #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))

MQTT SSL connection failed.

11.1.1.38 #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))

MQTT subscribe message creation failed.

11.1.1.39 #define QAPI_NET_MQTT_ERR_SUBSCRIBE_UNKNOWN_TOPIC ((qapi_-Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_E-RR_NUM_START + 21))

MQTT subscribe unknown topic.

11.1.1.40 #define QAPI_NET_MQTT_ERR_UNSUBSCRIBE_MSG_CREATION_-FAILED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 22))

MQTT unsubscribe message creation failed.

11.1.1.41 #define QAPI_NET_MQTT_ERR_UNEXPECTED_MSG ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 23))

MQTT unexpected message receivied.

11.1.1.42 #define QAPI_NET_MQTT_ERR_PARTIAL_SUBSCRIPTION_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 24))

MQTT subscription failed.

11.1.1.43 #define QAPI_NET_MQTT_ERR_RESTORE_FAILURE ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, QAPI_NET_MQTT_ERR_NUM_START + 25))

MQTT restore failed.

11.1.1.44 #define QAPI_NET_MQTT_ERR_MAX_NUMS 26

MQTT maximum error number.

11.1.1.45 #define QAPI_NET_NIPD_FLOW_SUSPENDED ((qapi_Status_t)__QAPI_ERROR(QAPI_MOD_NETWORKING, 27))

Non-IP data flow suspended.

11.1.1.46 #define QAPI_OK ((qapi_Status_t)(0))

Success.

11.1.1.47 #define QAPI_ERROR ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 1)))

General error.

11.1.1.48 #define QAPI_ERR_INVALID_PARAM ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 2)))

Invalid parameter.

11.1.1.49 #define QAPI_ERR_NO_MEMORY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 3)))

Memory allocation error.

11.1.1.50 #define QAPI_ERR_NO_RESOURCE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 4)))

Resource allocation error.

11.1.1.51 #define QAPI_ERR_BUSY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 6)))

Operation is busy.

11.1.1.52 #define QAPI_ERR_NO_ENTRY ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 7)))

Entry was not found.

11.1.1.53 #define QAPI_ERR_NOT_SUPPORTED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 8)))

Feature is not supported.

11.1.1.54 #define QAPI_ERR_TIMEOUT ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 9)))

Operation timed out.

11.1.1.55 #define QAPI_ERR_BOUNDS ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 10)))

Out of bounds.

11.1.1.56 #define QAPI_ERR_BAD_PAYLOAD ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 11)))

Bad payload.

11.1.1.57 #define QAPI_ERR_EXISTS ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 12)))

Entry already exists.

11.1.1.58 #define QAPI_ERR_NOT_INITIALIZED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 13)))

Unintialized.

11.1.1.59 #define QAPI_ERR_INVALID_STATE ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 13)))

Invalid state.

11.1.1.60 #define QAPI_ERR_API_DEPRACATED ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BASE, 14)))

QAPI function is deprecated.

12 System Drivers APIs

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

- GPIO Interrupt Controller APIs
- PMM APIs

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


qapi_Status_t          nStatus;
qapi_Instance_Handle_t pH;
uint32_t               nLoopCounter = 0;

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

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

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

```
}

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

12.1.1 Typedef Documentation

12.1.1.1 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.

12.1.1.2 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.

12.1.1.3 typedef void∗ qapi_Instance_Handle_t

GPIO interrupt handle definition.

12.1.2 Enumeration Type Documentation

12.1.2.1 enum qapi_GPIOINT_Trigger_e

GPIO interrupt trigger type enumeration for supported triggers.

Enumerator:

QAPI_GPIOINT_TRIGGER_LEVEL_HIGH_E Level triggered active high.
QAPI_GPIOINT_TRIGGER_LEVEL_LOW_E Level triggered active low.
QAPI_GPIOINT_TRIGGER_EDGE_RISING_E Rising edge triggered.
QAPI_GPIOINT_TRIGGER_EDGE_FALLING_E Falling edge triggered.
QAPI_GPIOINT_TRIGGER_EDGE_DUAL_E Dual edge triggered.

12.1.2.2 enum qapi_GPIOINT_Priority_e

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

Enumerator:

QAPI_GPIOINT_PRIO_HIGHEST_E Highest priority.
QAPI_GPIOINT_PRIO_HIGH_E Medium-high priority.
QAPI_GPIOINT_PRIO_MEDIUM_E Medium priority.
QAPI_GPIOINT_PRIO_LOW_E Medium-low priority.
QAPI_GPIOINT_PRIO_LOWEST_E Highest priority.

12.1.3 Function Documentation

12.1.3.1 qapi_Status_t qapi_GPIOINT_Register_Interrupt (qapi_Instance_Handle_t * *pH,* uint32_t *nGpio,* qapi_GPIOINT_CB_t *pfnCallback,* qapi_GPIOINT_Callback_Data_t *nData,* qapi_GPIOINT_Trigger_e *eTrigger,* qapi_GPIOINT_Priority_e *ePriority,* qbool_t *bNmi*)

Registers a callback for a GPIO interrupt.

Registers a callback function with the GPIO interrupt controller and enables the interrupt. This function configures and routes the interrupt accordingly, as well as enabling it in the underlying layers.

Parameters

in	*pH*	Input handle to the client context.
in	*nGpio*	GPIO number to configure for an interrupt.
in	*pfnCallback*	Callback function pointer.
in	*nData*	Callback data.
in	*eTrigger*	Trigger type for the interrupt.
in	*ePriority*	Priority enumeration to determine the configuration of the GPIO interrupt.
in	*bNmi*	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.

12.1.3.2 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	*pH*	Input handle to the client context.
in	*nGpio*	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.

12.1.3.3 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	*pH*	Input handle to the client context.
in	*nGpio*	GPIO number in which to set the trigger.
in	*eTrigger*	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.

12.1.3.4 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.

12.1.3.5 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.

12.1.3.6 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	*pH*	Input handle to the client context.
in	*nGpio*	GPIO number to trigger.

Returns

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

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

12.1.3.7 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	*pH*	Input handle to the client context.
in	*nGpio*	GPIO number to trigger.
out	*pbIsPending*	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.

12.2 PMM APIs

Modern SoCs pack a lot of functionality but are often pin-limited owing to their shrinking size. This limitation is overcome by incorporating hardware to flexibly mux several different functionalities on a given physical pin under software control.

This module exposes an interface allowing its clients to manage desired functionalities on a set of physical GPIO pins on the SoC. The most common usage of this interface is to configure pins for discrete inputs or outputs to implement handshakes with external peripherals, sensors, or actuators.

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

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

qapi_GPIO_ID_t     gpio_id;
qapi_TLMM_Config_t tlmm_config;
qapi_Status_t      status = QAPI_OK;

tlmm_config.pin = 13;
tlmm_config.func = 1                // Using the functionality tied to
                                    // pin mux value 1
tlmm_config.dir = QAPI_GPIO_INPUT_E;
tlmm_config.pull = QAPI_GPIO_PULL_DOWN_E;
tlmm_config.drive = QAPI_GPIO_2MA_E; // drive is for output pins, specify
                                    // the default here

status = qapi_TLMM_Get_Gpio_ID( &tlmm_config, &gpio_id);

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

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

12.2.1 Data Structure Documentation

12.2.1.1 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.

12.2.2 Typedef Documentation

12.2.2.1 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.

12.2.3 Enumeration Type Documentation

12.2.3.1 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.

12.2.3.2 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.

12.2.3.3 enum qapi_GPIO_Drive_t

GPIO pin drive strength.

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

Enumerator:

QAPI_GPIO_2MA_E Specify a 2 mA drive.
QAPI_GPIO_4MA_E Specify a 4 mA drive.
QAPI_GPIO_6MA_E Specify a 6 mA drive.
QAPI_GPIO_8MA_E Specify an 8 mA drive.
QAPI_GPIO_10MA_E Specify a 10 mA drive.
QAPI_GPIO_12MA_E Specify a 12 mA drive.
QAPI_GPIO_14MA_E Specify a 14 mA drive.
QAPI_GPIO_16MA_E Specify a 16 mA drive.

12.2.3.4 enum qapi_GPIO_Value_t

GPIO output state specification.

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

Enumerator:

QAPI_GPIO_LOW_VALUE_E Drive the output LOW.
QAPI_GPIO_HIGH_VALUE_E Drive the output HIGH.

12.2.4 Function Documentation

12.2.4.1 qapi_Status_t qapi_TLMM_Get_Gpio_ID (qapi_TLMM_Config_t * *qapi_TLMM_Config,* qapi_GPIO_ID_t * *qapi_GPIO_ID*)

Gets a unique access ID.

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

Parameters

in	*qapi_TLMM_Config*	Pointer to the pin configuration data.
in	*qapi_GPIO_ID*	Pointer to a location in which to store the access ID.

Returns

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

12.2.4.2 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	*qapi_TLMM_Config*	Pointer to pin configuration data.
in	*qapi_GPIO_ID*	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.

12.2.4.3 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	*qapi_GPIO_ID*	Pin access ID retrieved from the qapi_TLMM_Get_Gpio_ID() call.
in	*qapi_TLMM_Config*	Pin configuration to use.

Returns

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

12.2.4.4 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	*qapi_GPIO_ID*	Pin access ID retrieved from the qapi_TLMM_Get_Gpio_ID() call.
in	*pin*	SoC pin number to configure.
in	*value*	Output value.

Returns

QAPI_OK – Operation completed successfully.
QAPI_ERR – Operation failed.

12.2.4.5 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	*qapi_GPIO_ID*	Pin access ID retrieved from the qapi_TLMM_Get_Gpio_ID() call.
in	*pin*	SoC pin number to configure.
out	*qapi_GPIO_Value*	GIPO pin value.

Returns

QAPI_OK – Operation completed successfully.
QAPI_ERR – Operation failed.

13 Diagnostic Services Module

This chapter describes the diagnostic (Diag) services APIs.

13.1 QAPI Diag Services APIs

13.1.1 Define Documentation

13.1.1.1 #define QAPI_DIAGPKT_DISPATCH_TABLE_REGISTER(*xx_subsysid, xx_entry, inbuf, inbuf_len, outbuf, outbuf_len*)

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

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

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

Parameters:

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

Returns QAPI status; see QAPI Status Codes.

13.1.1.2 #define QAPI_DIAGPKT_DISPATCH_TABLE_REGISTER_V2_DELAY(*xx_cmdcode, xx_subsysid, xx_entry, inbuf, inbuf_len, outbuf, outbuf_len*)

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

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

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

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

Parameters:

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

Returns QAPI status; see QAPI Status Codes.

13.1.1.3 #define QAPI_MSG(*xx_ss_id, xx_ss_mask, xx_fmt, ...*)

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

Parameters:

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

Returns QAPI status; see QAPI Status Codes.

13.1.1.4 #define QAPI_MSG_SPRINTF(*xx_ss_id, xx_ss_mask, xx_fmt, ...*)

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

Parameters:

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

Returns QAPI status; see QAPI Status Codes.

13.1.2 Function Documentation

13.1.2.1 qapi_Status_t qapi_user_space_tbl_reg_append_proc (diagpkt_master_table_type * *tbl_ptr*, diagpkt_user_space_table_type * *user_space_tbl_ptr*)

Registers the user table given to the diagpkt master table and creates a new entry in diagpkt_user_space_table with user_space_tbl_ptr. Updates the port field of the master table entry with the index of its entry in diagpkt_user_space_table.

Parameters

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

Returns

QAPI status; see QAPI Status Codes.

13.1.2.2 qapi_Status_t qapi_diagpkt_get_next_delayed_rsp_id (uint16_t * *delayed_rsp_id*)

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

Parameters

in	*delayed_rsp_id*	Address of the variable that will be updated with the delayed response ID from Diag.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.3 qapi_Status_t qapi_diagpkt_commit (uint32_t * *outbuf,* uint32_t *rsp_len,* uint32_t *rsp_flag*)

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

Parameters

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

Returns

QAPI status; see QAPI Status Codes.

13.1.2.4 qapi_Status_t qapi_user_space_tbl_dereg (diagpkt_user_table_entry_type * *tbl_ptr*)

Deregisters a user table with Diag.

Parameters

in	*tbl_ptr*	Address of the user table that is to be deregistered with Diag.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.5 qapi_Status_t qapi_msg_send (const msg_const_type * *const_blk,* uint32_t *num_args,* ...)

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

Parameters

in	*const_blk*	Constant information stored for a message.
in	*num_args*	Number of arguments for the message.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.6 qapi_Status_t qapi_msg_sprintf (const msg_const_type * *const_blk,* uint32_t *num_args,* ...)

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

Parameters

in	*const_blk*	Constant information stored for a message.
in	*num_args*	Number of arguments for the message.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.7 qapi_Status_t qapi_log_submit (void * *ptr*)

Logs an accumlated log entry. Header contents must be assigned by the caller before calling this function. Therefore, qapi_log_set_code(), qapi_log_set_length(), and qapi_log_set_timestamp() must be used before this call.

Parameters

in	*ptr*	Pointer to the client-allocated log packet.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.8 qapi_Status_t qapi_log_set_length (void * *ptr,* log_code_type *length*)

Sets the length field in the specified log record.

Parameters

in	*ptr*	Pointer to the client-allocated log packet.
in	*length*	Length of the client-allocated log packet.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.9 qapi_Status_t qapi_log_set_code (void * *ptr,* log_code_type *code*)

Sets the code field in the specified log record.

Parameters

in	*ptr*	Pointer to the client-allocated log packet.
in	*code*	Log code of the client-allocated log packet.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.10 qapi_Status_t qapi_log_set_timestamp (void * *plog_hdr_ptr*)

Sets the timestamp field in the specified log record.

Parameters

in	*plog_hdr_ptr*	Pointer to the client-allocated log packet.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.11 qapi_Status_t qapi_log_status (log_code_type *code*)

Checks whether a particular code is enabled for logging.

Parameters

in	*code*	Log code of the client-allocated log packet.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.12 qapi_Status_t qapi_event_report (event_id_enum_type *event_id*)

Reports an event without a payload.

Parameters

in	*event_id*	Event ID of the event to be reported.

Returns

QAPI status; see QAPI Status Codes.

13.1.2.13 qapi_Status_t qapi_event_report_payload (event_id_enum_type *event_id,* uint8_t *length,* void * *data*)

Reports an event with a payload.

Parameters

in	*event_id*	Event ID of the event to be reported.
in	*length*	Length of the event to be reported.
in	*data*	Payload of the event to be reported.

Returns

QAPI status; see QAPI Status Codes.

14 Storage Module

This chapter describes the file system data types and APIs.

- File System Data Types
- File System APIs
- FTL Data Types and APIs

14.1 File System Data Types

14.1.1 Data Structure Documentation

14.1.1.1 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.

14.1.1.2 struct qapi_FS_Statvfs_Type_s

File system information, used in the qapi_FS_Statvfs() API.

Data fields

Type	Parameter	Description
unsigned long	f_bsize	Fundamental file system block size. Minimum allocations in the file system happen at this size.
uint32	f_blocks	Maximum possible number of blocks available in the entire file system.
uint32	f_bfree	Total number of free blocks.
uint32	f_bavail	Number of free blocks currently available.
uint32	f_files	Total number of file serial numbers.
uint32	f_ffree	Total number of free file serial numbers.
uint32	f_favail	Number of file serial numbers available.
unsigned long	f_fsid	File system ID; this varies depending on the implementation of the file system.

Type	Parameter	Description
unsigned long	f_flag	Bitmask of f_flag values.
unsigned long	f_namemax	Maximum length of the name part of the string for a file, directory, or symlink.
unsigned long	f_maxwrite	Maximum number of bytes that can be written in a single write call.
uint32	f_balloc	Blocks allocated in the general pool.
uint32	f_hardalloc	Hard allocation count. Resource intensive, so this is not usually computed.
unsigned long	f_pathmax	Maximum path length, excluding the trailing NULL. The unit here is in terms of character symbols. The number of bytes needed to represent a character will vary depending on the file name encoding scheme. For example, in a UTF8 encoding scheme, representing a single character could take anywhere between 1 to 4 bytes.
unsigned long	f_is_case_-sensitive	Set to 1 if Path is case sensitive.
unsigned long	f_is_case_-preserving	Set to 1 if Path is case preserved.
unsigned long	f_max_file_size	Maximum file size in the units determined by the member f_max_file_size_unit.
unsigned long	f_max_file_-size_unit	Unit type for f_max_file_size.
unsigned long	f_max_open_-files	This member tells how many files can be kept opened for one particular file system. However, there is a global limit on how many files can be kept opened simultaneously across all file systems, which is configured by QAPI_FS_MAX_DESCRIPTORS.
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.

14.1.1.3 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.

14.1.2 Enumeration Type Documentation

Flag bits to open a file.

Enumerator:

QAPI_FS_O_RDONLY_E Open for read only.
QAPI_FS_O_WRONLY_E Open for write only.
QAPI_FS_O_RDWR_E Open for read and write.
QAPI_FS_O_CREAT_E Create the file if it does not exist.
QAPI_FS_O_EXCL_E Fail if the file exists.
QAPI_FS_O_TRUNC_E Truncate the file to zero bytes on successful open.
QAPI_FS_O_APPEND_E All writes will self-seek to the end of the file before writing.

Mode bits to open a file.

Enumerator:

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).

Offset bits to seek a file.

Enumerator:

QAPI_FS_SEEK_SET_E Set to Offset.
QAPI_FS_SEEK_CUR_E Set to Offset + current position.
QAPI_FS_SEEK_END_E Set to Offset + file size.

14.1.2.1 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.

14.1.2.2 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.

14.2 File System APIs

14.2.1 Function Documentation

14.2.1.1 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	*Oflag*	Argument that describes how this file is to be opened. It contains one of the following values: • QAPI_FS_O_RDONLY_E – Open for read only. • QAPI_FS_O_WRONLY_E – Open for write only. • QAPI_FS_O_RDWR_E – Open for read and write. In addition, the following flags can be bitwise ORed with this value: • QAPI_FS_O_APPEND_E – All writes will self-seek to the end of the file before writing. • QAPI_FS_O_CREAT_E – Create the file if it does not exist. • QAPI_FS_O_TRUNC_E – Truncate the file to zero bytes on successful open. The following flags can be used to specify common ways of opening files: • QAPI_FS_O_CREAT_E | QAPI_FS_O_TRUNC_E – Normal for writing a file. Creates it if it does not exist. The resulting file is zero bytes long. • QAPI_FS_O_CREAT_E | QAPI_FS_O_EXCL_E – Creates a file but fails if it already exists.
in	*Mode*	If QAPI_FS_O_CREAT is a part of Oflag, a third argument (Mode) must be passed to qapi_FS_open() to define the file permissions given to the newly created file. If QAPI_FS_O_CREAT is not a part of flag, set Mode=0. One or more of the following permission bits can be ORed as the Mode parameter: • QAPI_FS_S_IRUSR_E – Read permission for a user • QAPI_FS_S_IWUSR_E – Write permission for a user • QAPI_FS_S_IXUSR_E – Execute permission for a user • QAPI_FS_S_IRGRP_E – Read permission for a group • QAPI_FS_S_IWGRP_E – Write permission for a group • QAPI_FS_S_IXGRP_E – Execute permission for a group • QAPI_FS_S_IROTH_E – Read permission for other • QAPI_FS_S_IWOTH_E – Write permission for other • QAPI_FS_S_IXOTH_E – Execute permission for other • QAPI_FS_S_ISUID_E – Set UID on execution • QAPI_FS_S_ISGID_E – Set GID on execution • QAPI_FS_S_ISVTX_E – Sticky bit (hidden attribute in FAT)

out	*Fd_ptr*	Address of the file descriptor variable. On success, the file descriptor of an opened file is written to it. On failure, the variable is set to -1.

Returns

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

- QAPI_ERR_EEXIST – Cannot create a file with the path name because another file with the same name exists and an exclusive open is requested or a special (exclusive) file with the same path name exists.
- QAPI_ERR_ENOENT – No entry for the path name is found, the file cannot be opened (and a new file is not created because the QAPI_FS_O_CREAT flag was not supplied).
- QAPI_ERR_EMFILE – Maximum number of open descriptors is exceeded.
- QAPI_ERR_EISDIR – Opening a file with a write flag (QAPI_FS_O_WRONLY or QAPI_FS_O_RDWR) was denied because a directory with the same name exists.
- QAPI_ERR_ENOSPC – No space is left on the device.
- QAPI_ERR_ENAMETOOLONG – File/directory name exceeded the NAME_MAX limit or the path name exceeded the Path_MAX limit, which is 1024 bytes. The maximum length of a full path name, not including a trailing '\0' character.
- QAPI_ERR_ENOMEM – No more dynamic memory is available.
- QAPI_ERR_ENODEV – The device does not exist.
- QAPI_ERR_ENOTDIR – The file could not be created under a path that is not a directory. Another possibility is an open with the QAPI_FS_O_CREAT flag tried to create a file in the ROM file system.
- QAPI_ERR_EINVAL – Invalid parameter or path.

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.2 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	*Path*	Path of the file that is to be opened.
in	*Oflag*	Argument that describes how this file should be opened. See qapi_FS_Open_With_Mode().
out	*Fd_ptr*	Address of the file descriptor variable. On success, the file descriptor of an opened file is written to it. On failure, the variable is set to -1.

Returns

See qapi_FS_Open_With_Mode().

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.3 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	*Fd*	File descriptor obtained via the qapi_FS_Open() function.
out	*Buf*	Buffer where the read bytes from the file will be stored.
in	*Count*	Number of bytes to read from the file.
out	*Bytes_Read_Ptr*	This is a return from the function with the number of bytes actually read.

Returns

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

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

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

Returns

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

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.5 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	*Fd*	File descriptor obtained via the qapi_FS_Open() function.

Returns

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

14.2.1.6 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	*Old_Path*	Path name before the rename operation.
in	*New_Path*	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 suceeded.
- QAPI_ERR_EISIR – The New_Path is a directory.
- QAPI_ERR_EXDEV – A rename operation accross different file systems is not permitted.
- QAPI_ERR_ENOTEMPTY – The Old_Path directory is not empty.

14.2.1.7 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	*Path*	Path of the file whose length is to be truncated.
in	*Length*	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.

14.2.1.8 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	*Fd*	File descriptor obtained via the qapi_FS_Open() API.
in	*Offset*	New offset of the file.
in	*Whence*	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	*Actual_Offset_Ptr*	Upon success, the resulting offset as bytes from the beginning of the file is filled in this parameter. On failure, the variable is set to -1.

Returns

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

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

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.9 qapi_FS_Status_t qapi_FS_Mk_Dir (const char * *Path*, qapi_FS_Mode_t *Mode*)

Creates a new directory.

Parameters

in	*Path*	Path for the directory.
in	*Mode*	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.

14.2.1.10 qapi_FS_Status_t qapi_FS_Rm_Dir (const char * *Path*)

Removes a directory. Only empty directories can be removed.

Parameters

in	*Path*	Path of the directory that is to be removed.

Returns

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

- QAPI_ERR_ENOTDIR – The parameter Path is not a directory.
- QAPI_ERR_ENOTEMPTY – The directory is not empty.
- QAPI_ERR_ETXTBSY – The directory is in use or open.
- QAPI_ERR_EINVAL – Invalid parameter.

14.2.1.11 qapi_FS_Status_t qapi_FS_Unlink (const char * *Path*)

Removes a link to a closed file.

If the link Count goes to zero, this will also remove the file. The qapi_FS_Unlink() API can be used on all file system objects except for directories. Use qapi_FS_Rm_Dir() for directories.

Note: The file must be closed for unlinking or removing. If it is open, the error QAPI_ERR_ETXTBSY is returned, indicating that the file is not closed.

Parameters

in	*Path*	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.

14.2.1.12 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	*Path*	File descriptor of the file.
out	*SBuf*	For information on what is returned in the structure, see struct qapi_FS_Stat_Type_s.

Returns

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

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.13 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	*Fd*	Handle to a file otained using the qapi_FS_Open() API.
out	*SBuf*	Information is returned in the structure qapi_FS_Stat_Type_s.

Returns

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

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.14 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	*Path*	Valid path of a file or directory on the mounted file system.
out	*SBuf*	Information is returned in the structure qapi_FS_Statvfs_Type_s.

Returns

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

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.15 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	*Path*	Valid path of the directory to iterate.
out	*handle*	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.

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.16 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	*Iter_Hdl*	Handle to directory obtained by the qapi_FS_Iter_Open() API.

out	*Iter_Entry*	Details about the next entry found is filled in
		struct qapi_FS_Dirent {
		char file_Path[QAPI_FS_NAME_MAX+1]
		struct qapi_FS_Stat_Type_s SBuf
		uint32 qapi_FS_D_Stats_Present;
		}

- file_Path – Name of the directory component
- SBuf – Information about the component; See qapi_FS_Stat_Type_s for information about this structure
- qapi_FS_D_Stats_Present – This is a bitmask for the above structure that defines which fields are filled when this this API is called.
  Bitmasks for qapi_FS_D_Stats_Present are defined as:

```
::QAPI_FS_DIRENT_HAS_ST_DEV       = (1 << 1)
::QAPI_FS_DIRENT_HAS_ST_INO       = (1 << 2)
::QAPI_FS_DIRENT_HAS_ST_Mode      = (1 << 3)
::QAPI_FS_DIRENT_HAS_ST_NLINK     = (1 << 4)
::QAPI_FS_DIRENT_HAS_ST_SIZE      = (1 << 5)
::QAPI_FS_DIRENT_HAS_ST_BLKSIZE   = (1 << 6)
::QAPI_FS_DIRENT_HAS_ST_BLOCKS    = (1 << 7)
::QAPI_FS_DIRENT_HAS_ST_ATIME     = (1 << 8)
::QAPI_FS_DIRENT_HAS_ST_MTIME     = (1 << 9)
::QAPI_FS_DIRENT_HAS_ST_CTIME     = (1 << 10)
::QAPI_FS_DIRENT_HAS_ST_RDEV      = (1 << 11)
::QAPI_FS_DIRENT_HAS_ST_UID       = (1 << 12)
::QAPI_FS_DIRENT_HAS_ST_GID       = (1 << 13)
```

Returns

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

Note: If this API is called from user space with MMU enabled, the parameters must be user space addresses, otherwise an invalid parameter error will be returned.

14.2.1.17 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	*Iter_Hdl*	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.

14.2.1.18 qapi_FS_Status_t qapi_FS_Last_Error (void)

Returns the last error that occured 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.

14.3 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.

14.3.1 Define Documentation

14.3.1.1 #define __QAPI_FTL_ERROR(x) ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BSP_FTL, x)))

Utility macro to define QAPI_FTL-specific error types.

14.3.1.2 #define QAPI_FTL_NOT_INIT __QAPI_FTL_ERROR(0)

Error returned if FTL APIs are called without calling FTL init first.

14.3.1.3 #define QAPI_FTL_OUT_OF_GOOD_BLOCKS __QAPI_FTL_ERROR(1)

Error returned if NAND is out of good blocks.

14.3.1.4 #define QAPI_FTL_ERR_UNKNOWN __QAPI_FTL_ERROR(2)

Error returned if registering QAPI handler fails.

14.3.1.5 #define QAPI_FTL_ERR_INVLD_ID __QAPI_FTL_ERROR(3)

Error returned if QAPI handler is called with an invalid ID.

14.3.2 Data Structure Documentation

14.3.2.1 struct qapi_FTL_info_t

Structure for storing information about a partition.

Data fields

Type	Parameter	Description
uint8_t	device_name	Device name string.
uint32_t	maker_id	Manufacturer ID.
uint32_t	device_id	Device ID.
uint32_t	lpa_count	Number of LPAs in the device.
uint32_t	lpa_size_in_-kbytes	LPA size in kB.
uint32_t	erase_block_-count	Number of eraseable units in the partition.
uint32_t	erase_block_-size_in_kbytes	Erase unit size in kB.

14.3.3 Typedef Documentation

14.3.3.1 typedef void* qapi_FTL_client_t

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

14.3.4 Function Documentation

14.3.4.1 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.

Note: If this API is called from user space with MMU enabled, the parameter "handle" must be a user space address, otherwise an invalid parameter error will be returned.

Parameters

in	*part_name*	Name of the partition the client wants to use.
out	*handle*	Handle that is passed to the client for further use. The client must pass the address of the pointer in which this handle is to be stored. If the return status is QAPI_OK, handle will contain the handle to the partition, which is used for any read or write operation on partition part_name.

Returns

- QAPI_ERR_INVALID_PARAM – handle or part_name is NULL, or part_name is invalid.
- QAPI_ERROR – An internal failure occured.
- QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.
- QAPI_OK – Success.

14.3.4.2 qapi_Status_t qapi_FTL_Close (qapi_FTL_client_t * *handle*)

Closes a partition once the client is done with it.

Note: If this API is called from user space with MMU enabled, the parameter "handle" must be a user space address, otherwise an invalid parameter error will be returned.

Parameters

in	*handle*	Handle of the partition to be closed.

Returns

- QAPI_ERR_INVALID_PARAM – handle is NULL.
- QAPI_ERROR – An internal failure occured.
- QAPI_OK – Success.

14.3.4.3 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.

Note: If this API is called from user space with MMU enabled, the parameter "handle" must be a user space address, otherwise an invalid parameter error will be returned.

Parameters

in	*handle*	Handle returned from qapi_FTL_Open().
out	*info*	Pointer to where the information is stored.

Returns

- QAPI_ERR_INVALID_PARAM – handle or info is NULL.
- QAPI_OK – Success.

14.3.4.4 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.

Note: If this API is called from user space with MMU enabled, the parameter "buffer" must be a user space address, otherwise an invalid parameter error will be returned.

Parameters

in	*handle*	Handle returned from qapi_FTL_Open().
in	*lpa*	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	*lpa_count*	Number of LPAs or pages to read.
out	*buffer*	Pointer to where the read data is stored. It should be an uncached, physically contiguous DMA-BLE.

Returns

- QAPI_ERR_INVALID_PARAM – handle or lpa is NULL.
- QAPI_ERROR – An internal failure occured.
- QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.
- QAPI_OK – Success.

14.3.4.5 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.

Note: If this API is called from user space with MMU enabled, the parameter "buffer" must be a user space address, otherwise an invalid parameter error will be returned.

Parameters

in	*handle*	Handle returned from qapi_FTL_Open().
in	*lpa*	Logical page address (or page number) where the data is to be written. The LPA is with respect to the start of the partition
in	*lpa_count*	Number of LPAs or pages to write.
in	*buffer*	Pointer to the buffer to which the data is to be written. It should be an uncached, physically contiguous DMA-BLE.

Returns

- QAPI_ERR_INVALID_PARAM – handle or lpa is NULL.
- QAPI_ERROR – An internal failure occured.
- QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.

- QAPI_OK – Success.

14.3.4.6 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	*handle*	Handle returned from qapi_FTL_Open().
in	*erase_block*	Start erase block.
in	*erase_block_count*	Number of blocks to be erased from Flash starting at erase_block.

Returns

- QAPI_ERR_INVALID_PARAM – handle is NULL.
- QAPI_ERROR – An internal failure occured.
- QAPI_FTL_OUT_OF_GOOD_BLOCKS – The partition is not usable.
- QAPI_OK – Success.

15 Wired Connectivity Module

This chapter describes the USB data types and APIs.

- USB Data Types
- USB APIs

15.1 USB Data Types

Type definitions for USB QAPIs.

15.1.1 Typedef Documentation

15.1.1.1 typedef void(∗ qapi_USB_App_Rx_Cb_t)(void)

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

15.1.1.2 typedef void(∗ qapi_USB_App_Disconnect_Cb_t)(void)

Client callback function type to provide a disconnect notification.

15.1.2 Enumeration Type Documentation

15.1.2.1 enum qapi_USB_Ioctl_Cmd_t

IOCTL command type.

Enumerator:

QAPI_USB_RX_CB_REG_E IOCTL command argument to register a client callback.
QAPI_USB_DISCONNECT_CB_REG_E Command argument to register a disconnect callback.

15.2 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.
*
* Clients can register a disconnect callback if they wish to be notified
* whenever USB is disconnected

 void* Buffer;
 uint16 Max_Size;
 void* Data_Ptr;
 uint16 Len;
 void Callback_func(void);
 void disconnect_cb(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 for getting notified about available data
 // for client
 status = qapi_USB_Ioctl(QAPI_USB_RX_CB_REG_E, (void*)Callback_func);
 if (status != QAPI_OK) { ... }

 // To register disconnect callback
 status = qapi_USB_Ioctl(QAPI_USB_DISCONNECT_CB_REG_E, (void*)disconnect_cb);
 if (status != QAPI_OK) { ... }
```

15.2.1 Function Documentation

15.2.1.1 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.

15.2.1.2 qapi_USB_Status_t qapi_USB_Read (void ** *Buffer,* uint16_t *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	*Buffer*	Buffer to where the data is to be copied.
in	*Max_Size*	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.

15.2.1.3 qapi_USB_Status_t qapi_USB_Write (void * *Data_Ptr,* uint16_t *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	*Data_Ptr*	Pointer to the data that the client wishes to send.
in	*Len*	Length of the data to be sent.

Returns

QAPI_OK on success, a -ve error code on failure.

15.2.1.4 qapi_USB_Status_t qapi_USB_Ioctl (qapi_USB_Ioctl_Cmd_t *Cmd,* void * *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	*Cmd*	Determines for what the IOCTL is called. Currently, only the purpose stated above is valid.
in	*Param*	Can change based on the command passed. For currently used commands, 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.

16 Buses Module

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

- I2C Master APIs
- SPI Master APIs
- UART APIs

16.1 I2C Master APIs

I2C is a 2-wire bus used to connect low speed peripherals to a processor or a microcontroller. Common I2C peripherals include touch screen controllers, accelerometers, gyros, and ambient light and temperature sensors.

The 2-wire bus comprises a data line, a clock line, and basic START, STOP, and acknowledge signals to drive transfers on the bus. An I2C peripheral is also referred to as an I2C slave. The processor or microcontroller implements the I2C master as defined in the I2C specification. This documentation provides the software interface to access the I2C master implementation.

```
//
// The code sample below demonstrates the use of this interface.
//

void sample (void)
{
  void *client_handle = NULL;
  uint32_t transferred1, transferred2;
  uint8_t buffer[4] = { 1, 2, 3, 4 };

  qapi_Status_t res = QAPI_OK;
  qapi_I2CM_Config_t config;
  qapi_I2CM_Descriptor_t desc[2];

  // Obtain a client specific connection handle to the i2c bus instance 1
  res = qapi_I2CM_Open (QAPI_I2CM_INSTANCE_001_E, &client_handle);

  // Configure the bus speed and slave address
  config.bus_Frequency_KHz = 400;
  config.slave_Address     = 0x36;
  config.SMBUS_Mode        = FALSE;

  // <S>  - START bit
  // <P>  - STOP  bit
  // <Sr> - Repeat Start bit
  // <A>  - Acknowledge bit
  // <N>  - Not-Acknowledge bit
  // <R>  - Read Transfer
  // <W>  - Write Transfer

  // Single write transfer of N bytes
  // <S><slave_address><W><A><data1><A><data2><A>...<dataN><A><P>
  desc[0].buffer      = buffer;
  desc[0].length      = 4;
  desc[0].transferred = &transferred1;
  desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE |
     QAPI_I2C_FLAG_STOP;
  res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 1,
     client_callback, NULL);

  // Single read transfer of N bytes
  // <S><slave_address><R><A><data1><A><data2><A>...<dataN><N><P>
  desc[0].buffer      = buffer;
  desc[0].length      = 4;
  desc[0].transferred = &transferred1;
  desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ  |
     QAPI_I2C_FLAG_STOP;
  res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 1,
     client_callback, NULL);
```

```
  // Read N bytes from a register 0x01 on a sensor device
  // <S><slave_address><W><A><0x01><A><S><slave_address><R><A>
  //                      <data1><A><data2><A>...<dataN><N><P>
  uint8_t reg = 0x01;
  desc[0].buffer      = &reg;
  desc[0].length      = 1;
  desc[0].transferred = &transferred1;
  desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE;

  desc[1].buffer      = buffer;
  desc[1].length      = 4;
  desc[1].transferred = &transferred2;
  desc[1].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ  |
     QAPI_I2C_FLAG_STOP;
  res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 2,
     client_callback, NULL);

  // Read N bytes from eeprom address 0x0102
  // <S><slave_address><W><A><0x01><A><0x02><A><S><slave_address><R><A>
  //                                <data1><A><data2><A>...<dataN><N><P>
  uint8_t reg[2] = { 0x01, 0x02 };
  desc[0].buffer      = reg;
  desc[0].length      = 2;
  desc[0].transferred = &transferred1;
  desc[0].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_WRITE;

  desc[1].buffer      = buffer;
  desc[1].length      = 4;
  desc[1].transferred = &transferred2;
  desc[1].flags       = QAPI_I2C_FLAG_START | QAPI_I2C_FLAG_READ  |
     QAPI_I2C_FLAG_STOP;
  res = qapi_I2CM_Transfer (client_handle, &config, &desc[0], 2,
     client_callback, NULL);

  // Close the connection handle to the i2c bus instance
  res = qapi_I2CM_Close (client_handle);
}

void client_callback (uint32_t status, void *ctxt)
{
  // Transfer completed
}
```

16.1.1 Define Documentation

16.1.1.1 #define QAPI_I2C_FLAG_START 0x00000001

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

16.1.1.2 #define QAPI_I2C_FLAG_STOP 0x00000002

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

16.1.1.3 #define QAPI_I2C_FLAG_WRITE 0x00000004

Must be set to indicate a WRITE transfer.

16.1.1.4 #define QAPI_I2C_FLAG_READ 0x00000008

Must be set to indicate a READ transfer.

16.1.1.5 #define QAPI_I2C_TRANSFER_MASK (QAPI_I2C_FLAG_WRITE | QAPI_I2C_-FLAG_READ)

Transfer types.

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

Verifies the validity of flags.

16.1.2 Data Structure Documentation

16.1.2.1 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.

16.1.2.2 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.

16.1.3 Typedef Documentation

16.1.3.1 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().

16.1.4 Enumeration Type Documentation

16.1.4.1 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.

16.1.5 Function Documentation

16.1.5.1 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	*instance*	I2C instance that the client intends to initialize; see qapi_I2CM_Instance_t for details.
out	*i2c_Handle*	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.

16.1.5.2 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	*i2c_Handle*	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.

16.1.5.3 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 mutliple transfers, it must use an array of descriptors of type qapi_I2CM_Descriptor_t instead of calling the API multiple times.

Parameters

in	*i2c_Handle*	Handle to the I2C instance.
in	*config*	Slave configuration. See qapi_I2CM_Config_t for details.
in	*desc*	I2C transfer descriptor. See qapi_I2CM_Descriptor_t for details. This can be an array of descriptors.
in	*num_Descriptors*	Number of descriptors in the descriptor array.
in	*CB_Function*	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	*CB_Parameter*	Context that the client passes here is returned as is in the callback function.

in	*delay_us*	A delay in microseconds that specifies the time to wait before starting the transfer.

Returns

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

16.1.5.4 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	*i2c_Handle*	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.

16.1.5.5 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	*i2c_Handle*	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.

16.2 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

16.2.1 Data Structure Documentation

16.2.1.1 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_C-S_Polarity_t	SPIM_CS_-Polarity	CS polarity type to be used for the SPI core.
qapi_SPIM_-Byte_Order_t	SPIM_-endianness	
uint8_t	SPIM_Bits_-Per_Word	Endian-ness type used for the SPI transfer. SPI bits per word; any value from 3 to 31.
uint8_t	SPIM_Slave_-Index	Slave index, beginning at 0 if mulitple 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_C-S_Mode_t	SPIM_CS_-Mode	CS mode to be used for the SPI core.
qbool_t	loopback_-Mode	Normally 0. If set, the SPI controller will enable Loopback mode; used primarily for testing.

16.2.1.2 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.

16.2.2 Typedef Documentation

16.2.2.1 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.

16.2.3 Enumeration Type Documentation

16.2.3.1 enum qapi_SPIM_Instance_t

SPI instance enumeration.

This enumeration lists the possible SPI instance indicating which HW SPI master is to be used for the current SPI transaction.

Enumerator:

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

16.2.3.2 enum qapi_SPIM_Shift_Mode_t

SPI phase type.

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

Enumerator:

QAPI_SPIM_MODE_0_E CPOL = 0, CPHA = 0.
QAPI_SPIM_MODE_1_E CPOL = 0, CPHA = 1.
QAPI_SPIM_MODE_2_E CPOL = 1, CPHA = 0.
QAPI_SPIM_MODE_3_E CPOL = 1, CPHA = 1.

16.2.3.3 enum qapi_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.

16.2.3.4 enum qapi_SPIM_Byte_Order_t

Order in which bytes from Tx/Rx buffer words are put on the bus.

Enumerator:

SPI_NATIVE Native.
SPI_LITTLE_ENDIAN Little Endian.
SPI_BIG_ENDIAN Big Endian (network).

16.2.3.5 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.

16.2.4 Function Documentation

16.2.4.1 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	*instance*	SPI instance specified by qapi_SPIM_Instance_t.
out	*spi_Handle*	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.

16.2.4.2 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	*spi_Handle*	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.

16.2.4.3 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	*spi_Handle*	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.

16.2.4.4 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	*spi_Handle*	Driver handle returned by qapi_SPIM_Open().
in	*config*	Pointer to the SPI configuration structure described by qapi_SPIM_Config_t.
in	*desc*	Pointer to the structure described by qapi_SPIM_Descriptor_t. The descriptor can have NULL Tx OR Rx buffers if a half duplex transfer is selected.
in	*num_Descriptors*	Number of descriptor pointers the client wants to process.
in	*c_Fn*	Callback function to be invoked when the SPI transfer completes succesfully or with an error.
in	*c_Ctxt*	Pointer to a client object that will be returned as an argument to c_Fn.
in	*get_timestamp*	Boolean variable to indicate whether a tranasaction timestamp needs to be provided; this is not supported for the QUPv2 version.

Returns

- QAPI_OK – SPI master was enabled successfully.
- QAPI_SPIM_ERROR_INVALID_PARAM – One or more invalid parameters.
- QAPI_SPIM_ERROR_QUP_STATE_INVALID – SPI or BAM hardware is in a bad state.
- QAPI_SPIM_ERROR_TRANSFER_TIMEOUT – Transfer timed out.

16.2.4.5 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	*spi_handle*	Driver handle returned by qapi_SPIM_Open().

Returns

- QAPI_OK – SPI driver closed successfully.

16.3 UART APIs

This section descibes the UART data types and APIs.

16.3.1 Data Structure Documentation

16.3.1.1 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.

16.3.1.2 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.

16.3.2 Typedef Documentation

16.3.2.1 typedef void* qapi_UART_Handle_t

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

16.3.2.2 typedef void(* qapi_UART_Callback_Fn_t)(uint32_t num_bytes, void *cb_data)

Transmit and receive operation callback type.

Parameters

in	*num_bytes*	Number of bytes.
out	*cb_data*	Pointer to the callback data.

16.3.3 Enumeration Type Documentation

16.3.3.1 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.

16.3.3.2 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.

16.3.3.3 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.

16.3.3.4 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.

16.3.3.5 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.

16.3.3.6 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.

16.3.4 Function Documentation

16.3.4.1 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	*handle*	UART handle provided by qapi_UART_Open().

Returns

- QAPI_OK – Port close was successful.
- QAPI_ERROR – Port close failed.

16.3.4.2 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	*handle*	UART handle.
in	*id*	ID of the port to be opened.
in	*config*	Structure that holds all configuration data.

Returns

- QAPI_OK – Port open was successful.
- QAPI_ERROR – Port open failed.

16.3.4.3 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	*handle*	UART handle provided by qapi_UART_Open().
in	*buf*	Buffer to be filled with data.
in	*buf_Size*	Buffer size. Must be ≥ 4 and a multiple of 4.
in	*cb_Data*	Callback data to be passed when rx_cb_isr is called during Rx completion.

Returns

- QAPI_OK – Queuing of the receive buffer was successful.
- QAPI_ERROR – Queuing of the receive buffer failed.

16.3.4.4 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	*handle*	UART handle provided by qapi_UART_Open().
in	*buf*	Buffer with data for transmit.
in	*bytes_To_Tx*	Bytes of data to transmit.
in	*cb_Data*	Callback data to be passed when tx_cb_isr is called during Tx completion.

Returns

- QAPI_OK – Queuing of the transmit buffer was successful.
- QAPI_ERROR – Queuing of the transmit buffer failed.

16.3.4.5 qapi_Status_t qapi_UART_Power_On (qapi_UART_Handle_t *UART_Handle*)

Enables the UART hardware resources for a UART transaction.

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

Parameters

in	*UART_Handle*	Driver handle returned by qapi_UART_Open().

Returns

- QAPI_OK – UART powered on successfully.
- QAPI_ERROR – UART power on failed.

16.3.4.6 qapi_Status_t qapi_UART_Power_Off (qapi_UART_Handle_t *UART_Handle*)

Disables the UART hardware resources for a UART transaction.

This function turns off all resources used by the UART master. This includes clocks, power resources, and GPIOs. This function should be called to put the UART master in its lowest possible power state.

Parameters

in	*UART_Handle*	Driver handle returned by qapi_UART_Open().

Returns

- QAPI_OK – UART powered off successfully.
- QAPI_ERROR – UART power off failed.

16.3.4.7 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	*handle*	Driver handle returned by qapi_UART_Open().
in	*ioctl_Command*	Pass the commands listed with qapi_UART_Ioctl_Command_e.
in	*ioctl_Param*	Pass the argument associated with qapi_UART_Ioctl_Command_e.

Returns

- QAPI_OK – UART IOCTL configuration was successfull.
- QAPI_ERROR – UART IOCTL configuration failed.

17 Location Module

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

- Location APIs

17.1 Location APIs

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

17.1.1 Data Structure Documentation

17.1.1.1 struct qapi_Location_t

Structure for location information.

Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of qapi_Location_t.
qapi_Location-_Flags_Mask_t	flags	Bitwise OR of qapi_Location_Flags_t.
uint64_t	timestamp	UTC timestamp for a location fix; milliseconds since Jan. 1, 1970.
double	latitude	Latitude in degrees.
double	longitude	Longitude in degrees.
double	altitude	Altitude in meters above the WGS 84 reference ellipsoid.
double	altitudeMean-SeaLevel	Altitude in meters with respect to mean sea level.
float	speed	Speed in meters per second.
float	bearing	Bearing in degrees; range: 0 to 360.
float	accuracy	Accuracy in meters.
float	vertical-Accuracy	Vertical accuracy in meters.
float	speedAccuracy	Speed accuracy in meters/second.
float	bearing-Accuracy	Bearing accuracy in degrees (0 to 359.999).

17.1.1.2 struct qapi_Gnss_Data_t

Structure for GNSS data information.

Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of qapi_Gnss_Data_t.
uint32_t	jammerInd	Jammer indication.

17.1.1.3 struct qapi_Location_Options_t

Structure for location options.

Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of qapi_Location_Options_t.

Type	Parameter	Description
uint32_t	minInterval	There are three different interpretations of this field, depending if minDistance is 0 or not: 1. Time-based tracking (minDistance = 0). minInterval is the minimum time interval in milliseconds that must elapse between final position reports. 2. Distance-based tracking (minDistance > 0). minInterval is the maximum time period in milliseconds after the minimum distance criteria has been met within which a location update must be provided. If set to 0, an ideal value will be assumed by the engine. 3. Batching. minInterval is the minimum time interval in milliseconds that must elapse between position reports.
uint32_t	minDistance	Minimum distance in meters that must be traversed between position reports. Setting this interval to 0 will be a pure time-based tracking/batching.

17.1.1.4 struct qapi_Geofence_Option_t

Structure for Geofence options.

Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of qapi_Geofence_Option_t.
qapi_Geofence_Breach_Mask_t	breachTypeMask	Bitwise OR of qapi_Geofence_Breach_Mask_Bits_t bits.
uint32_t	responsiveness	Specifies in milliseconds the user-defined rate of detection for a Geofence breach. This may impact the time lag between the actual breach event and when it is reported. The gap between the actual breach and the time it is reported depends on the user setting. The power implication is inversely proportional to the responsiveness value set by the user. The higher the responsiveness value, the lower the power implications, and vice-versa.
uint32_t	dwellTime	Dwell time is the time in milliseconds a user spends in the Geofence before a dwell event is sent.

17.1.1.5 struct qapi_Geofence_Info_t

Structure for Geofence information.

Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of qapi_Geofence_Info_t.
double	latitude	Latitude of the center of the Geofence in degrees.
double	longitude	Longitude of the center of the Geofence in degrees.
double	radius	Radius of the Geofence in meters.

17.1.1.6 struct qapi_Geofence_Breach_Notification_t

Structure for Geofence breach notification.

Data fields

Type	Parameter	Description
size_t	size	Size. Set to the size of qapi_Geofence_Breach_Notification_t.
size_t	count	Number of IDs in the array.
uint32_t *	ids	Array of IDs that have been breached.
qapi_Location_t	location	Location associated with a breach.
qapi_Geofence_Breach_t	type	Type of breach.
uint64_t	timestamp	Timestamp of the breach.

17.1.1.7 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_Collective_Response_Callback	collectiveResponseCb	Collective response callback is mandatory.
qapi_Tracking_Callback	trackingCb	Tracking callback is optional.
qapi_Batching_Callback	batchingCb	Batching callback is optional.
qapi_Geofence_Breach_Callback	geofenceBreachCb	Geofence breach callback is optional.
qapi_Single_Shot_Callback	singleShotCb	Single shot callback is optional.
qapi_Gnss_Data_Callback	gnssDataCb	GNSS data callback is optional.

17.1.2 Typedef Documentation

17.1.2.1 typedef void(∗ qapi_Capabilities_Callback)(qapi_Location_Capabilities_Mask_t capabilitiesMask)

Provides the capabilities of the system. It is called once after qapi_Loc_Init() is called.

Parameters

in	*capabilitiesMask*	Bitwise OR of qapi_Location_Capabilities_Mask_Bits_t.

Returns

None.

17.1.2.2 typedef void(∗ qapi_Response_Callback)(qapi_Location_Error_t err, uint32_t id)

Response callback, which is used by all tracking, batching and Single Shot APIs. It is called for every tracking, batching and single shot API.

Parameters

in	*err*	qapi_Location_Error_t associated with the request. If this is not QAPI_LOCATION_ERROR_SUCCESS then id is not valid.
in	*id*	ID to be associated with the request.

Returns

None.

17.1.2.3 typedef void(∗ qapi_Collective_Response_Callback)(size_t count, qapi_Location_Error_t ∗err, uint32_t ∗ids)

Collective response callback is used by Geofence APIs. It is called for every Geofence API call.

Parameters

in	*count*	Number of locations in arrays.
in	*err*	Array of qapi_Location_Error_t associated with the request.
in	*ids*	Array of IDs to be associated with the request.

Returns

None.

17.1.2.4 typedef void(∗ qapi_Tracking_Callback)(qapi_Location_t location)

Tracking callback used for the qapi_Loc_Start_Tracking() API. This is an optional function and can be NULL. It is called when delivering a location in a tracking session.

Parameters

in	*location*	Structure containing information related to the tracked location.

Returns

None.

17.1.2.5 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	*count*	Number of locations in an array.
in	*location*	Array of location structures containing information related to the batched locations.

Returns

None.

17.1.2.6 typedef void(∗ qapi_Geofence_Breach_Callback)(qapi_Geofence_Breach_Notification_t geofenceBreachNotification)

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	*geofenceBreach-Notification*	Breach notification information.

Returns

None.

17.1.2.7 typedef void(∗ qapi_Single_Shot_Callback)(qapi_Location_t location, qapi_Location_Error_t err)

Single shot callback used for the qapi_Loc_Get_Single_Shot() API. This is an optional function and can be NULL. It is called when delivering a location in a single shot session broadcasted to all clients, no matter if a session has started by client.

Parameters

in	*location*	Structure containing information related to the tracked location.
in	*err*	qapi_Location_Error_t associated with the request. This could be QAPI_LOCATION_ERROR_SUCCESS (location is valid) or QAPI_LOCATION_ERROR_TIMEOUT (a timeout occurred, location is not valid).

Returns

None.

17.1.2.8 typedef void(∗ qapi_Gnss_Data_Callback)(qapi_Gnss_Data_t gnssData)

GNSS data callback used for the qapi_Loc_Start_Get_Gnss_Data() API. This is an optional function and can be NULL. It is called when delivering a GNSS Data structure. The GNSS data structure contains useful information (e.g., jammer indication). This callback will be called every second.

Parameters

in	*gnssData*	Structure containing information related to the requested GNSS Data.

Returns

None.

17.1.2.9 typedef uint32_t qapi_loc_client_id

Location client identifier.

17.1.3 Enumeration Type Documentation

17.1.3.1 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.
QAPI_LOCATION_ERROR_GEOFENCES_AT_MAX Maximum number of geofences reached.
QAPI_LOCATION_ERROR_NOT_SUPPORTED Not supported.
QAPI_LOCATION_ERROR_TIMEOUT Timeout when asking single shot.

17.1.3.2 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.
QAPI_LOCATION_HAS_VERTICAL_ACCURACY_BIT Location has valid vertical accuracy.
QAPI_LOCATION_HAS_SPEED_ACCURACY_BIT Location has valid speed accuracy.
QAPI_LOCATION_HAS_BEARING_ACCURACY_BIT Location has valid bearing accuracy.

17.1.3.3 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.

17.1.3.4 enum qapi_Geofence_Breach_Mask_Bits_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.

17.1.3.5 enum qapi_Location_Capabilities_Mask_Bits_t

Flags to indicate the capabilities bit.

Enumerator:

QAPI_LOCATION_CAPABILITIES_TIME_BASED_TRACKING_BIT Capabilities time-based tracking bit.
QAPI_LOCATION_CAPABILITIES_TIME_BASED_BATCHING_BIT Capabilities time-based batching bit.
QAPI_LOCATION_CAPABILITIES_DISTANCE_BASED_TRACKING_BIT Capabilities distance-based tracking bit.
QAPI_LOCATION_CAPABILITIES_DISTANCE_BASED_BATCHING_BIT Capabilities distance-based batching bit.
QAPI_LOCATION_CAPABILITIES_GEOFENCE_BIT Capabilities Geofence bit.
QAPI_LOCATION_CAPABILITIES_GNSS_DATA_BIT Capabilities Geofence bit.

17.1.3.6 enum qapi_Gnss_Sv_t

Flags to indicate the constellation type.

Enumerator:

QAPI_GNSS_SV_TYPE_UNKNOWN Unknown.
QAPI_GNSS_SV_TYPE_GPS GPS.
QAPI_GNSS_SV_TYPE_SBAS SBAS.
QAPI_GNSS_SV_TYPE_GLONASS GLONASS.
QAPI_GNSS_SV_TYPE_QZSS QZSS.
QAPI_GNSS_SV_TYPE_BEIDOU BEIDOU.
QAPI_GNSS_SV_TYPE_GALILEO GALILEO.

17.1.4 Function Documentation

17.1.4.1 qapi_Location_Error_t qapi_Loc_Init (qapi_loc_client_id * *pClientId,* const qapi_Location_Callbacks_t * *pCallbacks*)

Initializes a location session and registers the callbacks.

Parameters

out	*pClientId*	Pointer to client ID type, where the unique identifier for this location client is returned.
in	*pCallbacks*	Pointer to the structure with the callback functions to be registered.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_CALLBACK_MISSING – One of the mandatory callback functions is missing.
QAPI_LOCATION_ERROR_GENERAL_FAILURE – There is an internal error.
QAPI_LOCATION_ERROR_ALREADY_STARTED – A location session has already been initialized.

17.1.4.2 qapi_Location_Error_t qapi_Loc_Deinit (qapi_loc_client_id *clientId*)

De-initializes a location session.

Parameters

in	*clientId*	Client identifier for the location client to be deinitialized.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.3 qapi_Location_Error_t qapi_Loc_Set_User_Buffer (qapi_loc_client_id *clientId*, uint8_t * *pUserBuffer*, size_t *bufferSize*)

Sets the user buffer to be used for sending back callback data.

Parameters

in	*clientId*	Client ID for which user buffer is to be set
in	*pUserBuffer*	User memory buffer to hold information passed in callbacks. Note that since buffer is shared by all the callbacks this has to be consumed at the user side before it can be used by another callback to avoid any potential race condition.
in	*bufferSize*	Size of user memory buffer to hold information passed in callbacks. This size should be large enough to accomodate the largest data size passed in a callback.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_GENERAL_FAILURE – There is an internal error.

17.1.4.4 qapi_Location_Error_t qapi_Loc_Start_Tracking (qapi_loc_client_id *clientId*, const 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. responseCb returns: QAPI_LOCATION_ERROR_SUCCESS if session was successfully started. QAPI_LOCATION_ERROR_ALREADY_STARTED if a qapi_Loc_Start_Tracking session is already in progress. QAPI_LOCATION_ERROR_CALLBACK_MISSING if no trackingCb was passed in qapi_Loc_Init(). QAPI_LOCATION_ERROR_INVALID_PARAMETER if pOptions parameter is invalid.

Parameters

in	*clientId*	Client identifier for the location client.
in	*pOptions*	Pointer to a structure containing the options: • minInterval – There are two different interpretations of this field, depending if minDistance is 0 or not: 1. Time-based tracking (minDistance = 0). minInterval is the minimum time interval in milliseconds that must elapse between final position reports. 2. Distance-based tracking (minDistance > 0). minInterval is the maximum time period in milliseconds after the minimum distance criteria has been met within which a location update must be provided. If set to 0, an ideal value will be assumed by the engine. • minDistance – Minimum distance in meters that must be traversed between position reports. Setting this interval to 0 will be a pure time-based tracking.
out	*pSessionId*	Pointer to the session ID to be returned.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.5 qapi_Location_Error_t qapi_Loc_Stop_Tracking (qapi_loc_client_id *clientId,* uint32_t *sessionId*)

Stops a tracking session associated with the id parameter responseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if successful. QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a tracking session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*sessionId*	ID of the session to be stopped.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.6 qapi_Location_Error_t qapi_Loc_Update_Tracking_Options (qapi_loc_client_id *clientId,* uint32_t *sessionId,* const qapi_Location_Options_t * *pOptions*)

Changes the location options of a tracking session associated with the id parameter. responseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if successful.
QAPI_LOCATION_ERROR_INVALID_PARAMETER if pOptions parameter is invalid.
QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a tracking session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*sessionId*	ID of the session to be changed.
in	*pOptions*	Pointer to a structure containing the options: • minInterval – There are two different interpretations of this field, depending if minDistance is 0 or not: 1. Time-based tracking (minDistance = 0). minInterval is the minimum time interval in milliseconds that must elapse between final position reports. 2. Distance-based tracking (minDistance > 0). minInterval is the maximum time period in milliseconds after the minimum distance criteria has been met within which a location update must be provided. If set to 0, an ideal value will be assumed by the engine. • minDistance – Minimum distance in meters that must be traversed between position reports. Setting this interval to 0 will be a pure time-based tracking.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.7 qapi_Location_Error_t qapi_Loc_Start_Batching (qapi_loc_client_id *clientId,* const 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. responseCb returns: QAPI_LOCATION_ERROR_SUCCESS if session was successfully started. QAPI_LOCATION_ERROR_ALREADY_STARTED if a qapi_Loc_Start_Batching session is already in progress. QAPI_LOCATION_ERROR_CALLBACK_MISSING if no batchingCb was passed in qapi_Loc_Init(). QAPI_LOCATION_ERROR_INVALID_PARAMETER if pOptions parameter is invalid. QAPI_LOCATION_ERROR_NOT_SUPPORTED if batching is not supported.

Parameters

in	*clientId*	Client identifier for the location client.
in	*pOptions*	Pointer to a structure containing the options: • minInterval – minInterval is the minimum time interval in milliseconds that must elapse between position reports. • minDistance – Minimum distance in meters that must be traversed between position reports.
out	*pSessionId*	Pointer to the session ID to be returned.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.8 qapi_Location_Error_t qapi_Loc_Stop_Batching (qapi_loc_client_id *clientId,* uint32_t *sessionId*)

Stops a batching session associated with the id parameter. responseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if successful. QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a batching session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*sessionId*	ID of the session to be stopped.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.9 qapi_Location_Error_t qapi_Loc_Update_Batching_Options (qapi_loc_client_id *clientId,* uint32_t *sessionId,* const qapi_Location_Options_t * *pOptions*)

Changes the location options of a batching session associated with the id parameter. responseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if successful.
QAPI_LOCATION_ERROR_INVALID_PARAMETER if pOptions parameter is invalid.
QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a batching session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*sessionId*	ID of the session to be changed.
in	*pOptions*	Pointer to a structure containing the options: • minInterval – minInterval is the minimum time interval in milliseconds that must elapse between position reports. • minDistance – Minimum distance in meters that must be traversed between position reports.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.10 qapi_Location_Error_t qapi_Loc_Get_Batched_Locations (qapi_loc_client_id *clientId*, uint32_t *sessionId*, 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. responseCb returns: QAPI_LOCATION_ERROR_SUCCESS if successful, will be followed by batchingCallback call. QAPI_LOCATION_ERROR_CALLBACK_MISSING if no batchingCb was passed in qapi_Loc_Init(). QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a batching session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*sessionId*	ID of the session for which the number of locations is requested.
in	*count*	Number of requested locations. The client can set this to MAXINT to get all the batched locations. If set to 0 no location will be present in the callback function.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.11 qapi_Location_Error_t qapi_Loc_Add_Geofences (qapi_loc_client_id *clientId*, size_t *count*, const qapi_Geofence_Option_t * *pOptions*, const qapi_Geofence_Info_t * *pInfo*, uint32_t ** *pIdArray*)

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 collective 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. collectiveResponseCb returns: QAPI_LOCATION_ERROR_SUCCESS if session was successful QAPI_LOCATION_ERROR_CALLBACK_MISSING if no geofenceBreachCb QAPI_LOCATION_ERROR_INVALID_PARAMETER if any parameters are invalid QAPI_LOCATION_ERROR_NOT_SUPPORTED if geofence is not supported.

Parameters

in	*clientId*	Client identifier for the location client.
in	*count*	Number of Geofences to be added.
in	*pOptions*	Array of structures containing the options: • breachTypeMask – Bitwise OR of GeofenceBreachTypeMask bits • responsiveness in milliseconds • dwellTime in seconds
in	*pInfo*	Array of structures containing the data: • Latitude of the center of the Geofence in degrees • Longitude of the center of the Geofence in degrees • Radius of the Geofence in meters

out	*pIdArray*	Array of IDs of Geofences to be returned.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.12 qapi_Location_Error_t qapi_Loc_Remove_Geofences (qapi_loc_client_id *clientId,* size_t *count,* const uint32_t * *pIDs*)

Removes a specified number of Geofences. collectiveResponseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if session was successful
QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a geofence session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*count*	Number of Geofences to be removed.
in	*pIDs*	Array of IDs of the Geofences to be removed.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.13 qapi_Location_Error_t qapi_Loc_Modify_Geofences (qapi_loc_client_id *clientId,* size_t *count,* const uint32_t * *pIDs,* const qapi_Geofence_Option_t * *options*)

Modifies a specified number of Geofences. collectiveResponseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if session was successful
QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a geofence session.
QAPI_LOCATION_ERROR_INVALID_PARAMETER if any parameters are invalid.

Parameters

in	*clientId*	Client identifier for the location client.
in	*count*	Number of Geofences to be modified.
in	*pIDs*	Array of IDs of the Geofences to be modified.
in	*options*	Array of structures containing the options: • breachTypeMask – Bitwise OR of GeofenceBreachTypeMask bits • responsiveness in milliseconds • dwellTime in seconds

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.

QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.14 qapi_Location_Error_t qapi_Loc_Pause_Geofences (qapi_loc_client_id *clientId*, size_t *count*, const 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. collectiveResponseCb returns: QAPI_LOCATION_ERROR_SUCCESS if session was successful QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a geofence session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*count*	Number of Geofences to be paused.
in	*pIDs*	Array of IDs of the Geofences to be paused.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.15 qapi_Location_Error_t qapi_Loc_Resume_Geofences (qapi_loc_client_id *clientId*, size_t *count*, const uint32_t * *pIDs*)

Resumes a specified number of Geofences that are paused. collectiveResponseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if session was successful
QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a geofence session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*count*	Number of Geofences to be resumed.
in	*pIDs*	Array of IDs of the Geofences to be resumed.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.16 qapi_Location_Error_t qapi_Loc_Get_Single_Shot (qapi_loc_client_id *clientId*, qapi_Location_Power_Level_t *powerLevel*, uint32_t * *pSessionId*)

Attempts a single location fix. It returns a session ID that will be used by qapi_Loc_Cancel_Single_Shot API and in the response callback to match the command with a response. responseCb returns:
QAPI_LOCATION_ERROR_SUCCESS if session was successfully started.
QAPI_LOCATION_ERROR_CALLBACK_MISSING if no singleShotCb passed in qapi_Loc_Init().
QAPI_LOCATION_ERROR_INVALID_PARAMETER if anyparameter is invalid. If responseCb reports LOCATION_ERROR_SUCESS, then the following is what can happen to end the single shot session: 1) A

location will be reported on the singleShotCb. 2) QAPI_LOCATION_ERROR_TIMEOUT will be reported on the singleShotCb. 3) The single shot session is canceled using the qapi_Loc_Cancel_Single_Shot API In either of these cases, the session is considered complete and the session id will no longer be valid.

Parameters

in	*clientId*	Client identifier for the location client.
in	*powerLevel*	Indicates what available technologies to use to compute the location.
out	*pSessionId*	Pointer to the session ID to be returned.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.17 qapi_Location_Error_t qapi_Loc_Cancel_Single_Shot (qapi_loc_client_id *clientId*, uint32_t *sessionId*)

Cancels a single shot session. responseCb returns: QAPI_LOCATION_ERROR_SUCCESS if successful. QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a single shot session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*sessionId*	ID of the single shot session to be cancelled.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.18 qapi_Location_Error_t qapi_Loc_Start_Get_Gnss_Data (qapi_loc_client_id *clientId*, uint32_t * *pSessionId*)

Starts a Get GNSS data session, which returns a session ID that will be used by the qapi_Loc_Stop_Get_Gnss_Data() API and in the response callback to match the command with a response. GNSS data is reported on the GNSS data callback passed in qapi_Loc_Init() periodically (every second until qapi_Loc_Stop_Get_Gnss_Data() is called).

responseCb returns:

QAPI_LOCATION_ERROR_SUCCESS if session was successfully started.

QAPI_LOCATION_ERROR_ALREADY_STARTED if a qapi_Loc_Start_Get_Gnss_Data() session is already in progress.

QAPI_LOCATION_ERROR_CALLBACK_MISSING if no gnssDataCb was passed in qapi_Loc_Init().

Parameters

in	*clientId*	Client identifier for the location client.
out	*pSessionId*	Pointer to the session ID to be returned.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

17.1.4.19 qapi_Location_Error_t qapi_Loc_Stop_Get_Gnss_Data (qapi_loc_client_id *clientId,* uint32_t *sessionId*)

Stops a Get GNSS data session associated with the ID parameter.

responseCb returns:

QAPI_LOCATION_ERROR_SUCCESS if successful.

QAPI_LOCATION_ERROR_ID_UNKNOWN if clientId is not associated with a Get GNSS data session.

Parameters

in	*clientId*	Client identifier for the location client.
in	*sessionId*	ID of the session to be stopped.

Returns

QAPI_LOCATION_ERROR_SUCCESS – The operation was successful.
QAPI_LOCATION_ERROR_NOT_INITIALIZED – No location session has been initialized.

18 Timer and Battery Modules

This chapter describes the timer and battery data types and APIs.

- Timer APIs
- PMIC RTC APIs
- PMIC Battery Status Information

18.1 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.

```
* The 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);
```

18.1.1 Data Structure Documentation

18.1.1.1 struct qapi_TIMER_define_attr_t

Timer define attribute type.

This type is used to specify parameters when defining a timer.

```
* sigs_func_ptr will depend on the value of qapi_TIMER_notify_t.
* qapi_TIMER_notify_t == QAPI_TIMER_NO_NOTIFY_TYPE,
* sigs_func_ptr = Don't care
*
* qapi_TIMER_notify_t == QAPI_TIMER_NATIVE_OS_SIGNAL_TYPE,
* sigs_func_ptr = qurt signal object
*
* qapi_TIMER_notify_t == QAPI_TIMER_FUNC1_CB_TYPE,
* sigs_func_ptr == specify a callback of type qapi_TIMER_cb_t
*
```

Data fields

Type	Parameter	Description
qbool_t	deferrable	FALSE = deferrable.
qapi_TIMER_-notify_t	cb_type	Type of notification to receive.
void *	sigs_func_ptr	Specify the signal object or callback function.
uint32_t	sigs_mask_data	Specify the signal mask or callback data.

18.1.1.2 struct qapi_TIMER_get_cbinfo_t

Type used to get a user space expired timer's callback information.

This type is used to get a user space expired timer's callback information.

```
* data = Specify the callback data for func_ptr,
* func_ptr = function pointer needs to be invoked.
*
```

Data fields

Type	Parameter	Description
void *	func_ptr	Specify the callback function.
uint32_t	data	Specify the callback data.

18.1.1.3 struct qapi_TIMER_set_attr_t

Type used to specify parameters when starting a timer.

Data fields

Type	Parameter	Description
uint64_t	time	Timer duration.
uint64_t	reload	Reload duration.
qapi_TIMER_-unit_type	unit	Specify units for timer duration.

18.1.1.4 struct qapi_TIMER_get_info_attr_t

Type used to get information for a given timer.

Data fields

Type	Parameter	Description
qapi_TIMER_-info_type	timer_info	Timer information type.
qapi_TIMER_-unit_type	unit	Specify units to use for return.

18.1.1.5 struct qapi_time_julian_type

Time in Julian format.

Data fields

Type	Parameter	Description
uint16_t	year	Year (1980 through 2100).
uint16_t	month	Month of the year (1 through 12).
uint16_t	day	Day of the month (1 through 31).
uint16_t	hour	Hour of the day (0 through 23).
uint16_t	minute	Minute of the hour (0 through 59).
uint16_t	second	Second of the minute (0 through 59).
uint16_t	day_of_week	Day of the week (0 through 6 or Monday through Sunday).

18.1.1.6 union qapi_time_get_t

Used to specify parameters when getting the time.

```
* Pointers depend on the value of qapi_time_unit_type.
* qapi_time_unit_type == QAPI_TIME_STAMP,
* time_ts = Of type qapi_time_type
*
* qapi_time_unit_type == QAPI_TIME_MSECS,
* time_msecs = Of type unint64_t
*
* qapi_time_unit_type == QAPI_TIME_SECS,
* time_secs = Of type unint64_t
*
* qapi_time_unit_type == QAPI_TIME_JULIAN,
* time_julian = Of type qapi_time_julian_type
```

Data fields

Type	Parameter	Description
qapi_time_type	time_ts	Specify the qapi_time_type variable pointer.
uint64_t	time_msecs	Variable for getting time in msec.
uint64_t	time_secs	Variable for getting time in sec.
qapi_time_-julian_type	time_julian	Variable for getting time in Julian.

18.1.2 Typedef Documentation

18.1.2.1 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.

18.1.2.2 typedef void(∗ qapi_TIMER_cb_t)(uint32_t data)

Timer callback type.

Timer callbacks should adhere to this signature.

18.1.2.3 typedef unsigned long qapi_qword[2]

Time type.

Native timestamp type.

18.1.3 Enumeration Type Documentation

18.1.3.1 enum qapi_TIMER_notify_t

Timer notification type.

Enumeration of the notifications available on timer expiry.

Enumerator:

QAPI_TIMER_NO_NOTIFY_TYPE No notification.
QAPI_TIMER_NATIVE_OS_SIGNAL_TYPE Signal an object.
QAPI_TIMER_FUNC1_CB_TYPE Call back a function.

18.1.3.2 enum qapi_TIMER_unit_type

Timer unit type.

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

Enumerator:

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

18.1.3.3 enum qapi_TIMER_info_type

Timer information type.

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

Enumerator:

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

specified units.

18.1.3.4 enum qapi_time_unit_type

Time unit type.

Enumeration of the types of time that can be obtained from time get QAPI.

Enumerator:

QAPI_TIME_STAMP Return the time in timestamp format.
QAPI_TIME_MSECS Return the time in millisecond format.
QAPI_TIME_SECS Return the time in second format.
QAPI_TIME_JULIAN Return the time in Julian calendar format.

18.1.4 Function Documentation

18.1.4.1 qapi_Status_t qapi_time_get (qapi_time_unit_type *time_get_unit*, qapi_time_get_t * *time*)

Gets the time in the specified format.

Parameters

in	*time_get_unit*	Unit in which to get the time.
in	*time*	Pointer to the qapi_time_get_t variable.

Returns

QAPI_OK on success, an error code on failure.

18.1.4.2 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	*timer_handle*	Handle to the timer.
in	*timer_attr*	Attributes for defining the timer.

Returns

QAPI_OK on success, an error code on failure.

Side effects

Calling this API causes memory allocation. Therefore, whenever the timer usage is done and not

required, qapi_Timer_Undef() must be called to release the memory, otherwise it will cause a memory leak.

18.1.4.3 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	*timer_handle*	Handle to the timer.
in	*timer_attr*	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.

18.1.4.4 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	*timer_handle*	Handle to the timer.
out	*timer_info*	Type of information needed from the timer.
out	*data*	Returned timer information.

Returns

QAPI_OK on success, an error code is returned on failure.

18.1.4.5 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	*timeout*	Specify the duration to block the thread.
in	*unit*	Specify the units of the duration.
in	*non_deferrable*	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.

18.1.4.6 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	*timer_handle*	Timer handle for which to undefine the timer.

Returns

QAPI_OK on success, an error code on failure

18.1.4.7 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	*timer_handle*	Timer handle for which to stop the timer.

Returns

QAPI_OK on success, an error code on failure.

18.2 PMIC RTC APIs

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

18.2.1 Data Structure Documentation

18.2.1.1 struct qapi_PM_Rtc_Julian_Type_t

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.

18.2.2 Enumeration Type Documentation

18.2.2.1 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.

18.2.2.2 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.

18.2.2.3 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.

18.2.3 Function Documentation

18.2.3.1 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.

18.2.3.2 qapi_Status_t qapi_PM_Set_Rtc_Display_Mode (qapi_PM_Rtc_Display_-Type_t *mode*)

Configures the real time clock display mode (24 or 12 hour mode). The RTC defaults to 24 hr mode on phone power up and remains so until it is set to 12 hr mode explicitly using qapi_PM_Set_Rtc_Display_Mode().

Parameters

in	*mode*	New RTC time display mode to be used. Valid values (see qapi_PM_Rtc_Display_Type_t): • 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 spported.
- QAPI_ERROR – Any other errors.

18.2.3.3 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	*qapi_current_time_-ptr*	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.

18.2.3.4 qapi_Status_t qapi_PM_Rtc_Alarm_RW_Cmd (qapi_PM_Rtc_Cmd_Type_t *cmd,* qapi_PM_Rtc_Alarm_Type_t *what_alarm,* qapi_PM_Rtc_Julian_Type_t * *qapi_alarm_time_ptr*)

Reads/writes the time and date from/to the PMIC RTC. The time/date format must be in 24 or 12 hr mode depending on in which mode the RTC was initialized. See the description of 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	*cmd*	Indicates whether to set or get the current time in the RTC. Valid values (see qapi_PM_Rtc_Cmd_Type_t): • QAPI_PM_RTC_SET_CMD_E • QAPI_PM_RTC_GET_CMD_E
in	*what_alarm*	Alarm type. See qapi_PM_Rtc_Alarm_Type_t.
in	*qapi_alarm_time_ptr*	Depending on the command, this function will use the structure qapi_PM_Rtc_Julian_Type_t pointer to update or return the alaram time in the RTC.

Note

day_of_week is not required for setting the current time, but it returns the correct information when retrieving time from the RTC.

Returns

Possible values (see qapi_Status_t):

- QAPI_OK – Operation succeeded.
- QAPI_ERR_INVALID_PARAM – Invalid parameter.
- QAPI_ERROR – Any other errors.

18.3 PMIC Battery Status Information

This module provides the definitions to get the battery status information.

18.3.1 Define Documentation

18.3.1.1 #define ___QAPI_ERROR_PMIC(x) ((qapi_Status_t)(__QAPI_ERROR(QAPI_MOD_BSP_PMIC, x)))

Error macros defined for QAPI errors.

18.3.1.2 #define QAPI_ERR_BATT_ABSENT ___QAPI_ERROR_PMIC(1)

Error macro for battery absent.

18.3.2 Enumeration Type Documentation

18.3.2.1 enum qapi_PM_Battery_Technology_t

PMIC battery technology.

Enumerator:

QAPI_PMIC_BAT_TECH_LI_ION_E Li-Ion battery.
QAPI_PMIC_BAT_TECH_LI_POLYMER_E Li-Polymer battery.

18.3.2.2 enum qapi_PM_Smb_Presence_t

PMIC SMB presence.

Enumerator:

QAPI_SMB_ABSENT_E SMB is present.
QAPI_SMB_PRESENT_E SMB is absent.

18.3.2.3 enum qapi_PM_Battery_Temperature_t

Battery temperature.

Enumerator:

QAPI_BAT_COLD_E Battery is cold.
QAPI_BAT_HOT_E Battery is hot.
QAPI_BAT_GOOD_E Battery temperature is good.

18.3.2.4 enum qapi_PM_Battery_Health_t

Battery health.

Enumerator:

QAPI_BAT_OV_E Battery is over voltage.
QAPI_BAT_UV_E Battery is under voltage.
QAPI_BAT_MISSING_E Battery is missing.
QAPI_BAT_GOOD_HEALTH_E Battery health is good.

18.3.2.5 enum qapi_PM_Battery_Chg_Status_t

Battery chargin status.

Enumerator:

QAPI_BAT_DISCHARGING_E Battery is discharging.
QAPI_BAT_CHARGING_E Battery is charging.

18.3.2.6 enum qapi_PM_Battery_Chg_Src_t

Battery charger source.

Enumerator:

QAPI_USB_CDP_E Charger type is USB_CDP.
QAPI_USB_DCP_E Charger type is USB_DCP.
QAPI_USB_SDP_E Charger type is USB_SDP.
QAPI_CHG_ABSENT_E Charger is absent.

18.3.3 Function Documentation

18.3.3.1 qapi_Status_t qapi_Pmapp_Vbatt_Get_Battery_Status (uint8 * *qapi_batt_status*)

Gets the battery charge percentage.

Parameters

out	*qapi_batt_status*	Buffer from which to get the battery charge percentage.

Returns

See qapi_Status_t. Possible values:

- QAPI_OK – Operation succeeded.
- QAPI_ERR_INVALID_PARAM – Invalid parameter.
- QAPI_ERR_NOT_SUPPORTED – Feature is not supported.
- QAPI_ERROR – Other errors.

18.3.3.2 qapi_Status_t qapi_Pmapp_Vbatt_Get_Battery_Health (qapi_PM_Battery_Health_t * *qapi_batt_health*)

Gets the battery health.

Parameters

out	*qapi_batt_health*	Buffer from which to get the battery health.

Returns

See qapi_Status_t. Possible values:

- QAPI_OK – Operation succeeded.
- QAPI_ERR_INVALID_PARAM – Invalid parameter.
- QAPI_ERR_NOT_SUPPORTED – Feature is not supported.
- QAPI_ERROR – Other errors.

18.3.3.3 qapi_Status_t qapi_Pmapp_Vbatt_Get_Battery_Temparature (qapi_PM_Battery_Temperature_t * *qapi_batt_temp*)

Gets the battery temparature.

Parameters

out	*qapi_batt_temp*	Buffer from which to get the battery temparature.

Returns

See qapi_Status_t. Possible values:

- QAPI_OK – Operation succeeded.
- QAPI_ERR_INVALID_PARAM – Invalid parameter.
- QAPI_ERR_NOT_SUPPORTED – Feature is not supported.
- QAPI_ERROR – Other errors.

18.3.3.4 qapi_Status_t qapi_Pmapp_Vbatt_Get_Battery_Technology (qapi_PM_Battery_Technology_t * *qapi_batt_tech*)

Gets the battery technology.

Parameters

out	*qapi_batt_tech*	Buffer from which to get the battery technology.

Returns

See qapi_Status_t. Possible values:

- QAPI_OK – Operation succeeded.
- QAPI_ERR_INVALID_PARAM – Invalid parameter.
- QAPI_ERR_NOT_SUPPORTED – Feature is not supported.
- QAPI_ERROR – Other errors.

18.3.3.5 qapi_Status_t qapi_Pmapp_Vbatt_Get_Battery_Charge_Status (qapi_PM_-Battery_Chg_Status_t * *qapi_charge_status*)

Gets the battery charging status.

Parameters

out	*qapi_charge_status*	Buffer from which to get the battery charging status.

Returns

See qapi_Status_t. Possible values:

- QAPI_OK – Operation succeeded.
- QAPI_ERR_INVALID_PARAM – Invalid parameter.
- QAPI_ERR_NOT_SUPPORTED – Feature is not supported.
- QAPI_ERROR – Other errors.

18.3.3.6 qapi_Status_t qapi_Pmapp_Vbatt_Get_Battery_Charger_Source (qapi_PM_Battery_Chg_Src_t * *qapi_charger_source*)

Gets the charger type.

Parameters

out	*qapi_charger_source*	Buffer from whichto get the battery charge source type, i.e., the charger type.

Returns

See qapi_Status_t. Possible values:

- QAPI_OK – Operation succeeded.
- QAPI_ERR_INVALID_PARAM – Invalid parameter.
- QAPI_ERR_NOT_SUPPORTED – Feature is not supported.
- QAPI_ERROR – Other errors.

19 Hardware Engine APIs

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

19.1 ADC Data Types

19.1.1 Define Documentation

19.1.1.1 #define ADC_INPUT_BATT_ID "BATT_ID"

Physical units are in millivolts.

19.1.1.2 #define ADC_INPUT_PA_THERM "PA_THERM"

Physical units are in degrees C.

19.1.1.3 #define ADC_INPUT_PA_THERM1 "PA_THERM1"

Physical units are in degrees C.

19.1.1.4 #define ADC_INPUT_PMIC_THERM "PMIC_THERM"

Physical units are in 0.001 gradients of degrees C.

19.1.1.5 #define ADC_INPUT_VBATT "VBATT"

Physical units are in millivolts.

19.1.1.6 #define ADC_INPUT_VPH_PWR "VPH_PWR"

Physical units are in millivolts.

19.1.1.7 #define ADC_INPUT_XO_THERM "XO_THERM"

Physical units are in 2^-10 degrees C.

19.1.1.8 #define ADC_INPUT_XO_THERM_GPS "XO_THERM_GPS"

Physical units are in 2^-10 degrees C.

19.1.2 Data Structure Documentation

19.1.2.1 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.

19.1.2.2 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.

19.1.2.3 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.

19.1.2.4 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.

19.1.2.5 struct qapi_ADC_Threshold_Result_t

ADC amplitude threshold result structure.

Data fields

Type	Parameter	Description
uint32_t	channel	Channel that was triggered.
qapi_ADC_Amp_Threshold_t	threshold	Threshold that was triggered.

19.1.2.6 struct qapi_ADC_Device_Properties_t

ADC device properties structure.

Data fields

Type	Parameter	Description
uint32_t	uNumChannels	Number of ADC channels.

19.1.2.7 struct qapi_AdcTM_Callback_Payload_Type_t

ADC TM callback payload structure

Data fields

Type	Parameter	Description
qapi_ADC_Amp_Threshold_t	eThresholdTriggered	Type of threshold that triggered.
uint32_t	uTMChannelIdx	TM channel index.
int32_t	nPhysicalTriggered	Physical value that triggered.

19.1.2.8 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.

19.1.2.9 struct qapi_AdcTM_Request_Params_Type_t

ADC TM request parameters structure.

Data fields

Type	Parameter	Description
qapi_Adc_Input_Properties_Type_t	adcTMInputProps	ADC channel input properties.
qapi_AdcTM_Threshold_Cb_Type	pfnAdcTMThresholdCb	Amplitude threshold callback type.
void *	pCtxt	Context specified when setting the threshold.

19.1.3 Typedef Documentation

19.1.3.1 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	*ctxt*	Context specified when setting the threshold.
in	*result*	Threshold crossing result.

Returns

None.

19.1.3.2 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	*ctxt*	Context specified when setting the threshold.
in	*result*	Threshold crossing result.

Returns

None.

19.1.4 Enumeration Type Documentation

19.1.4.1 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.

19.2 ADC APIs

The analog-to-digital converter (ADC) allows an analog signal to be sampled and digitally represented. The SoC features an on-die ADC that supports reading multiple channels. The ADC can perform single- shot and recurring measurements.

The ADC is configurable via static parameters. See the ADC tunable board file for the statically defined parameters.

This programming interface allows client software to configure channels, perform single readings, set a threshold if the channel is an ADC TM channel before reading the channel, and get ADC data samples. The code snippet below shows an example usage.

```
* The code snippet below demonstrates use of this interface. The example
* below opens ADC to obtain a handle, sets the thresholds if the channel
* is an ADC TM channel, reads each ADC channel, and then closes the handle.

 qapi_Status_t status;
 qapi_ADC_Handle_t handle;
 uint32_t num_channels;
 uint32_t channel;
 qapi_ADC_Read_Result_t result;
 const char Channel_Name;
 uint32_t Channel_Name_Size;
 qapi_AdcTM_Input_Properties_Type_t Properties_TM;
 qapi_Adc_Input_Properties_Type_t Properties;
 uint32_t Enable;
 const qapi_AdcTM_Request_Params_Type_t  ADC_TM_Params, TM_Params_Type;
 const int32 Lower_Tolerance, Higher_Tolerance, Threshold_Desired;
 qapi_ADC_Amp_Threshold_t Threshold_Type;
 qapi_AdcTM_Range_Type_t ADC_TM_Range;
 int32 TM_Threshold_Set;

 status = qapi_ADC_Open(&handle, Dummy);
 if (status != QAPI_OK) { ... }

 //To read ADC channels
 status=qapi_ADC_Get_Input_Properties(&handle, Channel_Name,
                              Channel_Name_Size, Properties);
 if (status != QAPI_OK) { ... }

 // To read and configure ADC TM channels
 status=qapi_ADC_TM_Get_Input_Properties(&handle, Channel_Name,
                              Channel_Name_Size, Properties_TM);
 if (status != QAPI_OK) { ... }
 else
 {
   status=qapi_ADC_Get_Range(&handle, channel, ADC_TM_Range);
   if (status != QAPI_OK) { ... }

   status=qapi_ADC_Set_Amp_Threshold(&handle, ADC_TM_Params,
            Threshold_Type, Threshold_Desired, TM_Threshold_Set):
   if (status != QAPI_OK) { ... }

   //Enable Thresholds (Enable = 1)
   status=qapi_ADC_TM_Enable_Thresholds(&handle, Enable, Threshold_Type);
   if (status != QAPI_OK) { ...

   status=qapi_ADC_TM_Set_Tolerance(&handle, TM_Params_Type_Ptr,
                                  Lower_Tolerance, Higher_Tolerance);
```

```
  if (status != QAPI_OK) { ... }

  //Disable Thresholds (Enable = 0)
  status=qapi_ADC_TM_Enable_Thresholds(&handle, Enable, Threshold_Type);
  if (status != QAPI_OK) { ... }
}

for (channel = 0; channel < num_channels; channel++)
{
   status = qapi_ADC_Read_Channel(handle, channel, &result);
   if (status != QAPI_OK) { ... }

   // result.microvolts contains the reading
}
status = qapi_ADC_Close(handle, false);
if (status != QAPI_OK) { ... }
handle = NULL;
```

19.2.1 Function Documentation

19.2.1.1 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	*Handle*	Pointer to an ADC handle.
in	*Attributes*	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.

19.2.1.2 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	*Handle*	Handle provided by qapi_ADC_Open().
in	*Channel_Name_Ptr*	Pointer to ADC channel name pointer.
in	*Channel_Name_Size*	Size of channel name string.
out	*Properties_Ptr*	ADC channel configuration.

Returns

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

19.2.1.3 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	*Handle*	Handle provided by qapi_ADC_Open().
in	*Input_Prop_Ptr*	Properties pointer of channel provided by qapi_ADC_Get_Input_Properties().
out	*Result_Ptr*	ADC reading result structure.

Returns

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

19.2.1.4 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	*Handle*	Handle provided by qapi_ADC_Open().
in	*Channel_Name_Ptr*	Pointer to the ADC TM channel name pointer.
in	*Channel_Name_Size*	Size of channel name string.
out	*Properties_Ptr*	ADC TM channel configuration.

Returns

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

19.2.1.5 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	*Handle*	Handle provided by qapi_ADC_Open().
in	*In_Properties_Ptr*	Properties pointer of the channel provided by qapi_ADC_TM_Get_Input_Properties().
out	*ADC_TM_Range_Ptr*	Pointer to the channel range.

Returns

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

19.2.1.6 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_t * *Threshold_Desired_Ptr,* int32_t * *TM_Threshold_Set_Ptr*)

Sets the threshold-related configuration for ADC TM channels.

The threshold event is triggered once when the threshold is crossed:

- ADC_TM_THRESHOLD_LOWER: current reading <= *Threshold_Desired_Ptr
- ADC_TM_THRESHOLD_HIGHER: current reading >= *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	*Handle*	Handle provided by qapi_ADC_Open().
in	*ADC_TM_Params_Ptr*	Pointer to the threshold parameters.
in	*Threshold_Type*	Type of threshold.
in	*Threshold_Desired_Ptr*	Pointer to desired threshold value.
out	*TM_Threshold_Set_Ptr*	Pointer to threshold value actually set.

Returns

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

19.2.1.7 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	*Handle*	Handle provided by qapi_ADC_Open().
in	*Enable*	Enable or disable thresholds.
in	*Threshold_Type*	Type of threshold.

Returns

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

19.2.1.8 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_t * *Lower_Tolerance*, const int32_t * *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	*Handle*	Handle provided by qapi_ADC_Open().
in	*TM_Params_Type*	Pointer to threshold configuration of ADCM TM channel.
in	*Lower_Tolerance*	Pointer to lower tolerance.
in	*Higher_Tolerance*	Pointer to higher tolerance.

Returns

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

19.2.1.9 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	*Handle*	Handle provided by qapi_ADC_Open().
in	*keep_enabled*	Reserved parameter.

Returns

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

19.3 TSENS Data Types

This section provides the type definitions for temperature sensor APIs.

19.3.1 Data Structure Documentation

19.3.1.1 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.
uint32_t	uSensor	Sensor that was triggered.
int32_t	nTriggeredDeg-C	Temperature value that was triggered.

19.3.1.2 struct qapi_TSENS_Result_t

TSENS temperature result structure.

Data fields

Type	Parameter	Description
int32_t	deg_c	Temperature in degrees Celsius.

19.3.2 Typedef Documentation

19.3.2.1 typedef void(∗ QAPI_Tsens_Threshold_Cb_Type)(void ∗pCtxt, const qapi_TSENS_CallbackPayloadType_t ∗pPayload)

TSENS callback function type.

19.3.2.2 typedef void∗ qapi_TSENS_Handle_t

TSENS handler type.

19.3.3 Enumeration Type Documentation

19.3.3.1 enum qapi_TSENS_ThresholdType_t

Enumeration of TSENS temperature thresholds.

Enumerator:

QAPIS_TSENS_THRESHOLD_LOWER Lower threshold.
QAPIS_TSENS_THRESHOLD_UPPER Upper threshold.
QAPIS_TSENS_NUM_THRESHOLDS Number of thresholds.

19.4 TSENS APIs

The temperature sensor is used to monitor the temperature of the SoC using on-die analog sensors.

This programming interface allows client software to read the temperature returned by each sensor. The code snippet below shows an example usage.

Consult hardware documentation for the placement of the sensors on the die.

```
* The code snippet below demonstates usage of this interface. The example
* below opens TSENS to obtain a handle, gets the number of sensors, sets
* temparature thresholds for each sensor, reads each sensor's
* temperature, and then closes the handle.

qapi_Status_t status;
qapi_TSENS_Handle_t handle;
uint32_t num_sensors;
uint32_t sensor;
qapi_TSENS_Result_t result;
qapi_TSENS_ThresholdType_t Threshold_Type;
int32_t Threshold_Degree;
QAPI_Tsens_Threshold_Cb_Type Threshold_CB;

status = qapi_TSENS_Open(&handle);
if (status != QAPI_OK) { ... }

status = qapi_TSENS_Get_Num_Sensors(handle, &num_sensors);
if (status != QAPI_OK) { ... }

for (sensor = 0; sensor < num_sensors; sensor++)
{
  status = qapi_TSENS_Get_Calibration_Status(handle, sensor, &result);
  if (status != QAPI_OK) { ... }

  else
     {
       status=qapi_TSENS_Get_Temp(handle, sensor, &result);
       if (status != QAPI_OK) { ... }

       else
          {
            status= qapi_TSENS_Set_Thresholds(handle, sensor,
                            Threshold_Type, Threshold_Degree,
                                    Threshold_CB, context_ptr);
              if (status != QAPI_OK) { ... }

          else
                     {
                status=qapi_TSENS_Set_Enable_Thresholds(handle,enable);
                      if (status != QAPI_OK) { ... }
                     }
           }
       }
   // result->Deg_C is the temperature in degrees Celsius
}
status = qapi_TSENS_Close(handle);
if (status != QAPI_OK) { ... }
handle = NULL;
```

19.4.1 Function Documentation

19.4.1.1 qapi_Status_t qapi_TSENS_Open (qapi_TSENS_Handle_t * *Handle*)

Opens TSENS.

Parameters

out	*Handle*	Pointer to a TSENS handle.

Returns

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

19.4.1.2 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	*Handle*	Handle provided by qapi_TSENS_Open().
out	*Num_Sensors_Ptr*	Number of sensors

Returns

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

19.4.1.3 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 hundreths 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	*Handle*	Handle provided by qapi_TSENS_Open().
in	*Sensor_Num*	Selected sensor
out	*Temp_Result_Ptr*	Temperature reported by the sensor.

Returns

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

19.4.1.4 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	*Handle*	Handle provided by qapi_TSENS_Open().
in	*Sensor_Num*	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.

19.4.1.5 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	*Handle*	Handle provided by qapi_TSENS_Open().
in	*Sensor_Num*	Selected sensor.
in	*Threshold_Type*	Threshold typeSelected sensor.
in	*Threshold_Degree*	Threshold in degrees centigrade.
in	*Threshold_CB*	Threshold callback.
in	*Context_Ptr*	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.

19.4.1.6 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	*Handle*	Handle provided by qapi_TSENS_Open().
in	*Enable_Threshold*	Enable or disable the threshold.

Returns

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

19.4.1.7 qapi_Status_t qapi_TSENS_Close (qapi_TSENS_Handle_t *Handle*)

Closes TSENS.

Parameters

in	*Handle*	Handle provided by qapi_TSENS_Open().

Returns

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

20 System Power Save Management

This chapter describes the system power save management (PSM) data types and APIs.

- PSM Data Types and Macros
- PSM APIs

20.1 PSM Data Types and Macros

This section provides PSM type definitions and macros.

PSM Client Status Messages

- #define QAPI_ERR_PSM_FAIL __QAPI_PSM_ERROR(1)
- #define QAPI_ERR_PSM_GENERIC_FAILURE __QAPI_PSM_ERROR(2)
- #define QAPI_ERR_PSM_APP_NOT_REGISTERED __QAPI_PSM_ERROR(3)
- #define QAPI_ERR_PSM_WRONG_ARGUMENTS __QAPI_PSM_ERROR(4)
- #define QAPI_ERR_PSM_IPC_FAILURE __QAPI_PSM_ERROR(5)
- #define QAPI_ERR_PSM_INVALID_ACTIVE_TIME __QAPI_PSM_ERROR(6)

20.1.1 Define Documentation

20.1.1.1 #define QAPI_ERR_PSM_FAIL __QAPI_PSM_ERROR(1)

Failure or invalid operation (unused).

20.1.1.2 #define QAPI_ERR_PSM_GENERIC_FAILURE __QAPI_PSM_ERROR(2)

Failure to send a request to the PSM Daemon.

20.1.1.3 #define QAPI_ERR_PSM_APP_NOT_REGISTERED __QAPI_PSM_ERROR(3)

The client ID passed is not a registered application.

20.1.1.4 #define QAPI_ERR_PSM_WRONG_ARGUMENTS __QAPI_PSM_ERROR(4)

NULL or invalid arguments were sent.

20.1.1.5 #define QAPI_ERR_PSM_IPC_FAILURE __QAPI_PSM_ERROR(5)

Internal failure to establish communication with the PSM Daemon.

20.1.1.6 #define QAPI_ERR_PSM_INVALID_ACTIVE_TIME __QAPI_PSM_ERROR(6)

An invalid active time was sent.

20.1.2 Data Structure Documentation

20.1.2.1 struct psm_time_info_type

PSM time information.

Data fields

Type	Parameter	Description
psm_time_-format_type_e	time_format_-flag	Time format. See psm_time_format_type_e.
pm_rtc_julian_-type	wakeup_time	Time in broken down format if the time_format_flag is set to PSM_TIME_IN_TM.
int	psm_duration_-in_secs	Time in seconds if the time_format_flag is set to PSM_TIME_IN_SECS.

20.1.2.2 struct psm_info_type

PSM information type.

Data fields

Type	Parameter	Description
int	active_time_in-_secs	Active time is the duration the PSM server must wait before entering PSM mode. The purpose of this time is to provide a chance for the MTC server to react.
psm_wakeup_-type_e	psm_wakeup_-type	Next wake up from PSM mode is for measurement purpose or measurement and network access.
psm_time_info-_type	psm_time_info	PSM time information. See psm_time_info_type.

20.1.2.3 struct psm_status_msg_type

PSM status message type.

Data fields

Type	Parameter	Description
int	client_id	Client ID.
int	status	PSM status. See psm_status_type_e.
int	reason	PSM reject reason. See psm_reject_reason_type_e.

20.1.3 Typedef Documentation

20.1.3.1 typedef void(∗ psm_client_cb_type)(psm_status_msg_type ∗)

PSM status callback type.

20.1.3.2 typedef void(∗ psm_util_timer_expiry_cb_type)(void ∗, size_t)

PSM timer expiry callback type.

20.1.4 Enumeration Type Documentation

20.1.4.1 enum psm_status_type_e

Enumeration of status types.

Enumerator:

PSM_STATUS_REJECT PSM enter request is rejected
PSM_STATUS_READY Ready to enter PSM mode.
PSM_STATUS_NOT_READY Not ready to enter PSM.
PSM_STATUS_COMPLETE Entered PSM mode; the system might shut down at any time.
PSM_STATUS_DISCONNECTED PSM server is down.
PSM_STATUS_MODEM_LOADED Modem is loaded as part of bootup.
PSM_STATUS_MODEM_NOT_LOADED Modem is not loaded as part of bootup.
PSM_STATUS_NW_OOS Network is OOS.
PSM_STATUS_NW_LIMITED_SERVICE Network is in Limited Service state.
PSM_STATUS_HEALTH_CHECK Application health check.
PSM_STATUS_FEATURE_ENABLED Feature is dynamically enabled.
PSM_STATUS_FEATURE_DISABLED Feature is dynamically disabled.

20.1.4.2 enum psm_reject_reason_type_e

Enumeration of reasons for rejection.

Enumerator:

PSM_REJECT_REASON_NONE No reject reason.
PSM_REJECT_REASON_NOT_ENABLED PSM feature is not enabled.
PSM_REJECT_REASON_MODEM_NOT_READY Modem is not ready to enter PSM mode.
PSM_REJECT_REASON_DURATION_TOO_SHORT PSM duration is too short to enter PSM mode.
PSM_REJECT_REASON_INCORRECT_USAGE PSM usage is wrong. Incorrect voting, re-vote before cancel, and other aspects will be rejected with this message.

20.1.4.3 enum psm_error_type_e

Enumeration of PSM error types.

Enumerator:

PSM_ERR_NONE Success.
PSM_ERR_FAIL Failure.
PSM_ERR_GENERIC_FAILURE Miscellaneous failure.
PSM_ERR_APP_NOT_REGISTERED Application is not registered with the PSM server.
PSM_ERR_WRONG_ARGUMENTS Wrong input arguments.
PSM_ERR_IPC_FAILURE Failure to communicate with the PSM server.
PSM_ERR_INVALID_ACTIVE_TIME Invalid active time value passed. Refer to 3GPP TS 24.008, Table 10.5.172 and Table 10.5.163a for valid active time values.

20.1.4.4 enum psm_time_format_type_e

PSM time format.

Enumerator:

PSM_TIME_IN_TM Specify time in broken down format.
PSM_TIME_IN_SECS Specify time in seconds.

20.1.4.5 enum psm_wakeup_type_e

PSM wakeup type.

Enumerator:

PSM_WAKEUP_MEASUREMENT_ONLY Next wake up from PSM is for measurement purpose only.
PSM_WAKEUP_MEASUREMENT_NW_ACCESS Next wake up from PSM is for measurement and network access.

20.2 PSM APIs

This section provides the PSM functions.

20.2.1 Function Documentation

20.2.1.1 qapi_Status_t qapi_PSM_Client_Register (int32_t * *client_id,* psm_client_cb_type *cb_func,* psm_status_msg_type * *cb_msg*)

Makes the application known to the PSM server as a PSM-aware application. This is the first API every PSM-aware application is to call. Every application that needs network related-functionality must call this API.

Registering a client enabled the PSM-aware application to vote for the PSM time and readiness when required. The callback is used by the PSM server to inform the application of all PSM events. A maximum of 20 clients can be registered at a time with server.

Parameters

out	*client_id*	Pointer to the stored ID (as an integer) of the registered client.
in	*cb_func*	Callback function of type psm_client_cb_type. The server invokes this function to notify the client of PSM events. PSM events contain status and reason. See psm_status_type_e and psm_reject_reason_type_e.
in	*cb_msg*	Callback message of type psm_status_msg_type passed when the sent callback is invoked. This message contains the ID, status, and reason. See psm_status_msg_type, psm_status_type_e, and psm_reject_reason_type_e.

Returns

Returns QAPI_OK on success or a -ve error code on failure.

- QAPI_ERR_PSM_WRONG_ARGUMENTS – One or more of the arguments are invalid or NULL.
- QAPI_ERR_PSM_GENERIC_FAILURE – Registration failed because the maximum client limit of 20 was exceeded.
- QAPI_ERR_ESPIPE – Some file descriptors (like pipes and FIFOs) are not seekable.

20.2.1.2 qapi_Status_t qapi_PSM_Client_Unregister (int32_t *client_id*)

Unregisters the PSM-aware application with the PSM server. Callbacks registered with the server by the application will no longer be used to send any messages by the server.

Unregistered applications cannot vote for PSM. Reregistration can be done using the qapi_PSM_Client_Register() call. Unregistered PSM-aware applications should be prepared for device shutdown without any further information.

Parameters

in	*client_id*	Client ID obtained during registration.

Returns

Returns QAPI_OK on success or a -ve error code on failure.

- QAPI_ERR_PSM_APP_NOT_REGISTERED – Invalid client ID.
- QAPI_ERR_PSM_GENERIC_FAILURE – Communication with the server failed.

20.2.1.3 qapi_Status_t qapi_PSM_Client_Enter_Psm (int32_t *client_id,* psm_info_type ∗ *psm_info*)

Used by the application to indicate its intent to enter PSM mode.

The application must pass active_time in seconds, time in PSM mode, and whether the next wake up is for measurement purposes or access to the network. PSM time can be accepted in either broken down format or in seconds. A PSM-aware application blocks PSM entry if this API is not called indefinitely.

Parameters

in	*client_id*	Client ID obtained during registration.
in	*psm_info*	Pointer to a psm_info_type structure consisting of active time, the next wakeup time (time in PSM), and the next wakeup type. Based on the wakeup type, the server decides whether to load the modem as part of bootup.

Returns

Returns QAPI_OK on success or a -ve error code on failure.

- QAPI_ERR_PSM_WRONG_ARGUMENTS – One or more of the arguments are invalid or NULL.
- QAPI_ERR_PSM_APP_NOT_REGISTERED – Invalid client ID.
- QAPI_ERR_PSM_GENERIC_FAILURE – Communication with the server failed.

20.2.1.4 qapi_Status_t qapi_PSM_Client_Enter_Backoff (int32_t *client_id*)

Used by the application to indicate its intent to enter PSM mode due to a network out-of-service state or if the MTC server is not reachable. The MTC server refers to any entity with which the PSM client tries to communicate over the network.

The duration for which the application wants to enter PSM mode is decided by the PSM server based on the NV item configuration NV73784 (psm_duration_due_to_oos). In a case where there is no PSM-aware application registered, the server sets the device to the PSM state independently. PSM aware can even decide to use the Enter PSM API with the intended time on recieving such status indications.

A call to backoff overwrites any valid PSM vote of the same client. Calling backoff when there is no network out-of-service will be treated as a vote to PSM with a default time as set in NV73784.

Parameters

in	*client_id*	Client ID obtained during registration.

Returns

Returns QAPI_OK on success or a -ve error code on failure.

- QAPI_ERR_PSM_APP_NOT_REGISTERED – Invalid client ID.
- QAPI_ERR_PSM_GENERIC_FAILURE – Communication with the server failed.

20.2.1.5 qapi_Status_t qapi_PSM_Client_Cancel_Psm (int32_t *client_id*)

Cancels a previous request to enter PSM.

Parameters

in	*client_id*	Client ID obtained during registration.

Returns

Returns QAPI_OK on success or a -ve error code on failure.

- QAPI_ERR_PSM_APP_NOT_REGISTERED – Invalid client ID.
- QAPI_ERR_PSM_GENERIC_FAILURE – Communication with the server failed.

20.2.1.6 qapi_Status_t qapi_PSM_Client_Load_Modem (int32_t *client_id*)

Requests the PSM server to load the modem if it is not already loaded (PIL-based flavors only).

PSM-aware applications can load the modem dynamically based on the use case to save power. Applications are informed through the callback of the modem loading success/failure. Further, applications can vote for modem loading in the next bootup through the qapi_PSM_Client_Enter_Psm() call.

Parameters

in	*client_id*	Client ID obtained during registration.

Returns

Returns QAPI_OK on success or a -ve error code on failure.

- QAPI_ERR_PSM_APP_NOT_REGISTERED – Invalid client ID.
- QAPI_ERR_PSM_GENERIC_FAILURE – Communication with the server failed.

20.2.1.7 qapi_Status_t qapi_PSM_Client_Hc_Ack (int32_t *client_id*)

Application health check acknowledge API. PSM-aware applications must call this API every time it receives a PSM_STATUS_HEALTH_CHECK event.

This API ensures that every registered PSM-aware application is alive and functioning, and not stuck in a deadlock situation. Periodically, the PSM server uses the callback to send a PSM_STATUS_HEALTH_CHECK event. The application must call this API to acknowledge that the application is working. On failing to respond to Health Check, the application is treated as a dead application and the server votes for PSM on behalf of the dead application.

Time in PSM is as configured in NV setting NV73784 (psm_duration_due_to_oos).

Parameters

in	*client_id*	Client ID obtained during registration.

Returns

Returns QAPI_OK on success or a -ve error code on failure.

- QAPI_ERR_PSM_APP_NOT_REGISTERED – Invalid client ID.
- QAPI_ERR_PSM_GENERIC_FAILURE – Communication with the server failed.

21 Device Information Module

This chapter describes the device information data types and APIs.

21.1 Device Information

21.1.1 Define Documentation

21.1.1.1 #define QAPI_DEVICE_INFO_BUF_SIZE 128

Maximum size of qapi_Device_Info_t valuebuf.

21.1.2 Data Structure Documentation

21.1.2.1 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	Union of values.

21.1.2.2 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.

21.1.2.3 struct qapi_Device_Info_t.u.valuebuf

Data fields

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

21.1.3 Enumeration Type Documentation

21.1.3.1 enum qapi_Device_Info_ID_t

Device information types.

Enumerator:

QAPI_DEVICE_INFO_BUILD_ID_E Device BUILD_ID.
QAPI_DEVICE_INFO_IMEI_E Device IMEI.
QAPI_DEVICE_INFO_IMSI_E UIM IMSI.
QAPI_DEVICE_INFO_OS_VERSION_E Device OS version.
QAPI_DEVICE_INFO_MANUFACTURER_E Device manufacturer.
QAPI_DEVICE_INFO_MODEL_ID_E Device model ID.
QAPI_DEVICE_INFO_BATTERY_STATUS_E Device battery status.
QAPI_DEVICE_INFO_BATTERY_PERCENTAGE_E Device battery percentage.
QAPI_DEIVCE_INFO_TIME_ZONE_E Device time zone.
QAPI_DEIVCE_INFO_ICCID_E Device ICCID.
QAPI_DEVICE_INFO_4G_SIG_STRENGTH_E Network signal strength.
QAPI_DEVICE_INFO_BASE_STATION_ID_E Network base station ID.
QAPI_DEVICE_INFO_MCC_E Network MCC.
QAPI_DEVICE_INFO_MNC_E Network MNC.
QAPI_DEVICE_INFO_SERVICE_STATE_E Network service status.
QAPI_DEVICE_INFO_MDN_E Device MDN.
QAPI_DEVICE_INFO_TAC_E Network tracking area code.
QAPI_DEVICE_INFO_CELL_ID_E Network cell ID.
QAPI_DEVICE_INFO_RCCS_E Network RRC state.
QAPI_DEVICE_INFO_EMMS_E Network EMM state.
QAPI_DEVICE_INFO_SERVING_PCI_E Network serving cell PCI.
QAPI_DEVICE_INFO_SERVING_RSRQ_E Serving cell RSRQ.
QAPI_DEVICE_INFO_SERVING_EARFCN_E Serving cell EARFCN.
QAPI_DEVICE_INFO_NETWORK_IND_E Network indication.
QAPI_DEVICE_INFO_ROAMING_E Roaming status.
QAPI_DEVICE_INFO_LAST_POWER_ON_E Last power on time.
QAPI_DEVICE_INFO_CHIPID_STRING_E Chipset name.
QAPI_DEVICE_INFO_APN_PROFILE_INDEX_E APN profile index.
QAPI_DEVICE_INFO_SIM_STATE_E SIM state.
QAPI_DEVICE_INFO_NETWORK_BEARER_E Network bearer.
QAPI_DEVICE_INFO_LINK_QUALITY_E Network link quality.
QAPI_DEVICE_INFO_TX_BYTES_E Device Tx bytes.
QAPI_DEVICE_INFO_RX_BYTES_E Device Rx bytes.
QAPI_DEVICE_INFO_ANY Any device information.

21.1.3.2 enum qapi_Device_Info_Type_t

Device information response types.

Enumerator:

QAPI_DEVICE_INFO_TYPE_BOOLEAN_E Response type is Boolean.
QAPI_DEVICE_INFO_TYPE_INTEGER_E Response type is integer.
QAPI_DEVICE_INFO_TYPE_BUFFER_E Response type is buffer.

21.1.4 Function Documentation

21.1.4.1 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.

21.1.4.2 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	*id*	Information ID.
out	*info*	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.

21.1.4.3 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	*id*	Information ID.
in	*callback*	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.

21.1.4.4 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.

21.1.4.5 qapi_Status_t qapi_Device_Info_Reset (void)

Resets the device.

Returns

QAPI_OK on success, QAPI_ERROR on failure.

22 LWM2M APIs

This chapter describes the Light Weight Machine to Machine (LWM2M) data types and APIs.

- LWM2M Data Types
- LWM2M APIs

22.1 LWM2M Data Types

This section provides the LWM2M data types.

LWM2M Client Request IDs

- #define **TXM_QAPI_LWM2M_REGISTER_APP** TXM_QAPI_NET_LWM2M_BASE + 1
- #define **TXM_QAPI_LWM2M_REGISTER_APP_EXTENDED** TXM_QAPI_NET_LWM2M_BASE + 2
- #define **TXM_QAPI_LWM2M_DEREGISTER_APP** TXM_QAPI_NET_LWM2M_BASE + 3
- #define **TXM_QAPI_LWM2M_OBSERVE** TXM_QAPI_NET_LWM2M_BASE + 4
- #define **TXM_QAPI_LWM2M_CANCEL_OBSERVE** TXM_QAPI_NET_LWM2M_BASE + 5
- #define **TXM_QAPI_LWM2M_CREATE_OBJECT_INSTANCE** TXM_QAPI_NET_LWM2M_BASE + 6
- #define **TXM_QAPI_LWM2M_DELETE_OBJECT_INSTANCE** TXM_QAPI_NET_LWM2M_BASE + 7
- #define **TXM_QAPI_LWM2M_GET** TXM_QAPI_NET_LWM2M_BASE + 8
- #define **TXM_QAPI_LWM2M_SET** TXM_QAPI_NET_LWM2M_BASE + 9
- #define **TXM_QAPI_LW2M_SEND_MESSAGE** TXM_QAPI_NET_LWM2M_BASE + 10
- #define **TXM_QAPI_LWM2M_ENCODE_APP_PAYLOAD** TXM_QAPI_NET_LWM2M_BASE + 11
- #define **TXM_QAPI_LWM2M_DECODE_APP_PAYLOAD** TXM_QAPI_NET_LWM2M_BASE + 12
- #define **TXM_QAPI_LWM2M_WAKEUP** TXM_QAPI_NET_LWM2M_BASE + 13
- #define **TXM_QAPI_LWM2M_CONFIG_CLIENT** TXM_QAPI_NET_LWM2M_BASE + 14
- #define **TXM_QAPI_LWM2M_DEFAULT_ATTR_INFO** TXM_QAPI_NET_LWM2M_BASE + 15
- #define **TXM_QAPI_LWM2M_SET_SRV_LIFETIME** TXM_QAPI_NET_LWM2M_BASE + 16
- #define **TXM_QAPI_LWM2M_GET_SRV_LIFETIME** TXM_QAPI_NET_LWM2M_BASE + 17
- #define **TXM_QAPI_LWM2M_ENCODE_DATA** TXM_QAPI_NET_LWM2M_BASE + 18
- #define **TXM_QAPI_LWM2M_DECODE_DATA** TXM_QAPI_NET_LWM2M_BASE + 19

22.1.1 Define Documentation

22.1.1.1 #define QAPI_LWM2M_SERVER_ID_INFO(*msg_buf, msg_len, server_id*)

Value:

```
{                                                                \
  server_id = 0x00;                                              \
  if (msg_len)                                                   \
    server_id = *((uint16_t *)(msg_buf + (msg_len - 2)));      \
}
```

Retrieve the LWM2M server short ID from the message ID information.

22.1.1.2 #define qapi_Net_LWM2M_Pass_Pool_Ptr(*a, b*) lwm2m_update_byte_pool(a, b)

Macro that passes a Byte Pool pointer for the LWM2M application.

Parameter a – Handle.

Parameter b – Pointer to the Byte Pool.

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

Note: This macro is only used in the DAM space.

22.1.1.3 #define qapi_Net_LWM2M_DeRegister_App(*a*) lwm2m_destroy_indirection(a, TXM_QAPI_LWM2M_DEREGISTER_APP)

Macro that releases a Byte Pool pointer for the LWM2M application.

Parameter a – Handle.

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

Note: This macro is only used in the DAM space.

22.1.2 Data Structure Documentation

22.1.2.1 struct qapi_Net_LWM2M_Id_Info_t

Structure to indicate the object/instance/resource ID for which the application is interested in monitoring or getting the value.

Data fields

Type	Parameter	Description
struct qapi_Net_LWM2M_Id_Info_s *	next	Pointer to the next ID information.
uint8_t	id_set	ID category defined in qapi_lwm2m_id.
uint16_t	object_ID	Object ID.
uint8_t	instance_ID	Object instance ID.
uint8_t	resource_ID	Resource ID.

22.1.2.2 struct qapi_Net_LWM2M_Object_Info_t

Structure to indicate the object/instance/resource for which the application is interested in monitoring or getting the value.

Data fields

Type	Parameter	Description
uint8_t	no_object_info	Number of object information blocks.
qapi_Net_LWM2M_Id_Info_t *	id_info	Pointer to the ID information.

22.1.2.3 struct qapi_Net_LWM2M_Flat_Data_t

LWM2M resource information (in flat format) to encode/decode data payload.

Data fields

Type	Parameter	Description
qapi_Net_LWM2M_Value_Type_t	type	Value type.
uint16_t	id	Resource ID.
union qapi_Net_LWM2M_Flat_Data_t	value	Union of value types.

22.1.2.4 union qapi_Net_LWM2M_Flat_Data_t.value

Union of value types.

Data fields

Type	Parameter	Description
bool	asBoolean	Value in boolean format.
int64_t	asInteger	Value as an integer.
double	asFloat	Value as a floating point.
objlink_t	asObjLink	Value as a object link.
value	asBuffer	Value as a string.
value	asChildren	Value as children.

22.1.2.5 struct qapi_Net_LWM2M_Flat_Data_t.value.asBuffer

Data fields

Type	Parameter	Description
size_t	length	String length.

Type	Parameter	Description
uint8_t *	buffer	Pointer to the string buffer.
uint8_t	block1_more	More blocks to be received.
uint32_t	block1_num	Block number
uint16_t	block1_size	Block size
uint32_t	block1_offset	Block offset

22.1.2.6 struct qapi_Net_LWM2M_Flat_Data_t.value.asChildren

Data fields

Type	Parameter	Description
size_t	count	Count of the children.
struct qapi_Net_LWM2M_Flat_Data_s *	array	Flat data array.

22.1.2.7 struct qapi_Net_LWM2M_Resource_Info_t

Structure that indicates the resource information that is to be created.

Data fields

Type	Parameter	Description
struct qapi_Net_LWM2M_Resource_Info_s *	next	Pointer to the next resource information.
uint16_t	resource_ID	Resource ID.
qapi_Net_LWM2M_Value_Type_t	type	Type of resource.
union qapi_Net_LWM2M_Resource_Info_t	value	Union of resource values.

22.1.2.8 union qapi_Net_LWM2M_Resource_Info_t.value

Union of resource values.

Data fields

Type	Parameter	Description
bool	asBoolean	Value in Boolean format.
int64_t	asInteger	Value as an integer.
double	asFloat	Value as a floating point.
objlink_t	asObjLink	Value as a object link.

Type	Parameter	Description
value	asBuffer	Value as a string.
value	asChildren	Value as a multi-resource instance

22.1.2.9 struct qapi_Net_LWM2M_Resource_Info_t.value.asBuffer

Data fields

Type	Parameter	Description
size_t	length	String length.
uint8_t *	buffer	Pointer to the string buffer.
uint8_t	block1_more	More blocks to be received.
uint32_t	block1_num	Block number
uint16_t	block1_size	Block size
uint32_t	block1_offset	Block offset

22.1.2.10 struct qapi_Net_LWM2M_Resource_Info_t.value.asChildren

Data fields

Type	Parameter	Description
size_t	count	Number of resources in the array.
qapi_Net_LWM2M_Flat_Data_t *	array	Array of resources.

22.1.2.11 struct qapi_Net_LWM2M_Instance_Info_t

Structure to indicate the instance information that is to be created.

Data fields

Type	Parameter	Description
struct qapi_Net_LWM2M_Instance_Info_s *	next	Pointer to the next object instance.
uint8_t	instance_ID	Instance ID.
uint8_t	no_resources	Number of resources.
qapi_Net_LWM2M_Resource_Info_t *	resource_info	Pointer to the resource information.

22.1.2.12 struct qapi_Net_LWM2M_Data_t

Structure that is populated by the application and provided to an LWM2M client when the application wants to create an instance of the LWM2M object to perform set and get operations.

Data fields

Type	Parameter	Description
struct qapi_Net-_LWM2M_-Data_s *	next	Pointer to the next object data.
uint16_t	object_ID	Object ID.
uint8_t	no_instances	Number of instances.
qapi_Net_LW-M2M_Instance-_Info_t *	instance_info	Pointer to the instance information.

22.1.2.13 struct qapi_Net_LWM2M_Obj_Info_t

LWM2M object/URI-related information.

Data fields

Type	Parameter	Description
qapi_Net_LW-M2M_ID_t	obj_mask	Bitmap indicating valid object fields.
uint16_t	obj_id	Object ID.
uint16_t	obj_inst_id	Object instance ID.
uint16_t	res_id	Resource ID.
uint16_t	res_inst_id	Resource instance ID.

22.1.2.14 struct qapi_Net_LWM2M_Attributes_t

LWM2M write attribute information.

Data fields

Type	Parameter	Description
qapi_Net_L-WM2M_Obj_-Info_t	obj_info	LWM2M object information associated with write attributes.
qapi_Net_LW-M2M_Write_-Attr_t	set_attr_mask	Bitmap indicating valid attribute fields to set.
qapi_Net_LW-M2M_Write_-Attr_t	clr_attr_mask	Bitmap indicating attribute fields to clear.
uint8_t	dim	Dimension.
uint32_t	minPeriod	Minimum period.

Type	Parameter	Description
uint32_t	maxPeriod	Maximum period.
double	greaterThan	Greater than.
double	lessThan	Less than.
uint8_t	step_valid	Step validity.
double	step	Step.
struct qapi_Net_LWM2M_Attributes_s *	next	Pointer to the next attributes information.

22.1.2.15 struct qapi_Net_LWM2M_Server_Data_t

LWM2M server request message data and internal LWM2M client state information.

Data fields

Type	Parameter	Description
qapi_Net_LWM2M_DL_Msg_t	msg_type	DL message type (requests, acknowledgements, or internal).
qapi_Net_LWM2M_Obj_Info_t	obj_info	Object information.
uint8_t	msg_id_len	Message ID length.
uint8_t	msg_id	Message ID. The message ID is transparent to the application, but is passed to the application for every message received from the server. The expectation is that the application stores the message ID associated with the message and passes it to the LWM2M client when a response or notification must be sent to the server. After the transaction pertaining to the message is complete, the message ID can be discarded from the application.
uint16_t	notification_id	Notification ID. When a notification is sent using qapi_Net_LWM2M_Send_Message(), the notification ID associated with the message is returned to the caller. It is the caller's responsibility to maintain the notification ID for observation mapping. Later, when the network does a Cancel Observation for a particular notification using RESET, it is indicated using the notification ID to the caller. Using this notification ID, the caller can cancel the observation. If the cancel observation was not using RESET, obj_info should have the information based on the observation that is to be cancelled.
qapi_Net_LWM2M_Content_Type_t	content_type	Current encoded data payload content type.
uint32_t	payload_len	Encoded data payload length.
uint8_t *	payload	Encoded data payload.

Type	Parameter	Description
qapi_Net_-LWM2M_-Attributes_t *	lwm2m_attr	Write attributes.
qapi_Net_LW-M2M_Event_t	event	Internal events.

22.1.2.16 struct qapi_Net_LWM2M_App_Ex_Obj_Data_t

LWM2M application response message data and notification-related information.

Data fields

Type	Parameter	Description
qapi_Net_L-WM2M_UL_-Msg_t	msg_type	UL message type (response or notification).
qapi_Net_L-WM2M_Obj_-Info_t	obj_info	Object information.
qapi_Net_-LWM2M_-Response_-Code_t	status_code	Application message status (applicable for responses only).
uint8_t	conf_msg	Confirmable (ACK) or nonconfirmable application response/notifications.
uint8_t	msg_id_len	Message ID length.
uint8_t	msg_id	Message ID. The message ID is transparent to the application, but is passed to the application for every message received from the server. The expectation is that the application stores the message ID associated with the message and passes it to the LWM2M client when a response or notification must be sent to the server. After the transaction pertaining to the message is complete, the message ID can be discarded from the application.
uint32_t	observation_-seq_num	Observation sequence number.
uint16_t	notification_id	Notification ID. When a notification is sent using qapi_Net_LWM2M_Send_-Message(), the notification ID associated with the message is returned to the caller. It is the caller's responsibility to maintain the notification ID for observation mapping. Later, when the network does a Cancel Observation for a particular notification using RESET, it is indicated using the notification ID to the caller. Using this notification ID, the caller can cancel the observation. If the cancel observation was not using RESET, obj_info should have the information based on the observation that is to be cancelled.

Type	Parameter	Description
qapi_Net_LWM2M_Content_Type_t	content_type	Encoded data payload content type.
uint32_t	payload_len	Encoded data payload length.
uint8_t *	payload	Encoded data payload.

22.1.2.17 struct qapi_Net_LWM2M_Config_Data_t

LWM2M config message data.

Data fields

Type	Parameter	Description
struct qapi_Net_LWM2M_Config_Data_s *	next	Pointer to the next object data.
qapi_Net_LWM2M_Config_Type_t	config_type	Configuration type.
union qapi_Net_LWM2M_Config_Data_t	value	Union of values.

22.1.2.18 union qapi_Net_LWM2M_Config_Data_t.value

Data fields

Type	Parameter	Description
bool	asBoolean	Present as a Boolean value.
int64_t	asInteger	Present as an integer value.
double	asFloat	Present as a float value.
value	asBuffer	Present as a buffer.

22.1.2.19 struct qapi_Net_LWM2M_Config_Data_t.value.asBuffer

Data fields

Type	Parameter	Description
size_t	length	Length of the buffer.
uint8_t *	buffer	Pointer to the buffer.

22.1.3 Typedef Documentation

22.1.3.1 typedef void* qapi_Net_LWM2M_App_Handler_t

Handler provide by LWM2M client to the application.

22.1.4 Enumeration Type Documentation

22.1.4.1 enum qapi_Net_LWM2M_Object_ID_t

Enum used to identify a particular object with an object ID.

Enumerator:

QAPI_NET_LWM2M_DEVICE_OBJECT_ID_E Device object ID.
QAPI_NET_LWM2M_FIRMWARE_UPDATE_OBJECT_ID_E Firmware update object ID.
QAPI_NET_LWM2M_LOCATION_OBJECT_ID_E Location object ID.
QAPI_NET_LWM2M_SOFTWARE_MGNT_OBJECT_ID_E Software management object ID.
QAPI_NET_LWM2M_DEVICE_CAP_OBJECT_ID_E Device capability object ID.

22.1.4.2 enum qapi_Net_LWM2M_Devicecap_Resource_Id_t

Enum used to identify a particular resource of a device capability object.

Enumerator:

QAPI_NET_LWM2M_DEVICE_RES_M_PROPERTY_E Property resource.
QAPI_NET_LWM2M_DEVICE_RES_M_GROUP_E Group resource.
QAPI_NET_LWM2M_DEVICE_RES_O_DESCRIPTION_E Description resource.
QAPI_NET_LWM2M_DEVICE_RES_O_ATTACHED_E Attached resource.
QAPI_NET_LWM2M_DEVICE_RES_M_ENABLED_E Enabled resource.
QAPI_NET_LWM2M_DEVICE_RES_M_OP_ENABLE_E Operation enable.
QAPI_NET_LWM2M_DEVICE_RES_M_OP_DISBALE_E Operation disable.
QAPI_NET_LWM2M_DEVICE_RES_O_NOTIFY_EN_E Notify EN ??.

22.1.4.3 enum qapi_Net_LWM2M_Fota_Resource_Id_t

Enum to identify valid firmware update resource IDs.

Enumerator:

QAPI_NET_LWM2M_FOTA_RES_M_PACKAGE_E Package resource.
QAPI_NET_LWM2M_FOTA_RES_M_PACKAGE_URI_E Package URI resource.
QAPI_NET_LWM2M_FOTA_RES_M_UPDATE_E Update resource.
QAPI_NET_LWM2M_FOTA_RES_M_STATE_E State resource.
QAPI_NET_LWM2M_FOTA_RES_M_UPDATE_RESULT_E Update result resource.
QAPI_NET_LWM2M_FOTA_RES_O_PACKAGE_NAME_E Package name resource.
QAPI_NET_LWM2M_FOTA_RES_O_PACKAGE_VERSION_E Package version resource.
QAPI_NET_LWM2M_FOTA_RES_O_UPDATE_PROTOCOL_SUPPORT_E Update protocol support resource.
QAPI_NET_LWM2M_FOTA_RES_M_UPDATE_DELIVERY_METHOD_E Update delivery method resource.

22.1.4.4 enum qapi_Net_LWM2M_Fota_Result_t

Enum to identify valid firmware update results.

Enumerator:

QAPI_NET_LWM2M_FOTA_RESULT_INITIAL_E Initial result.
QAPI_NET_LWM2M_FOTA_RESULT_UPDATE_SUCCESS_E Update success.
QAPI_NET_LWM2M_FOTA_RESULT_NOT_ENOUGH_STORAGE_E Not enough storage.
QAPI_NET_LWM2M_FOTA_RESULT_OUT_OF_MEMORY_E Out of memory.
QAPI_NET_LWM2M_FOTA_RESULT_CONNECTION_LOST_E Connection was lost.
QAPI_NET_LWM2M_FOTA_RESULT_CRC_CHECK_FAIL_E CRC check failed.
QAPI_NET_LWM2M_FOTA_RESULT_UNSUPPORTED_PACKAGE_TYPE_E Unsupported package type.
QAPI_NET_LWM2M_FOTA_RESULT_INVAILD_URI_E Invalid URI.
QAPI_NET_LWM2M_FOTA_RESULT_UPDATE_FAILED_E Update failed.
QAPI_NET_LWM2M_FOTA_RESULT_UNSUPPORTED_PROTOCOL_E Unsupported protocol.

22.1.4.5 enum qapi_Net_LWM2M_Fota_Supported_Protocols_t

Enum to identify supported protocols.

Enumerator:

QAPI_NET_LWM2M_FOTA_PROTOCOL_COAP COAP Protocol.
QAPI_NET_LWM2M_FOTA_PROTOCOL_COAPS COAPS Protocol.
QAPI_NET_LWM2M_FOTA_PROTOCOL_HTTP HTTP Protocol.
QAPI_NET_LWM2M_FOTA_PROTOCOL_HTTPS HTTPS Protocol.

22.1.4.6 enum qapi_Net_LWM2M_Fota_Update_Delivery_Method_t

Enum to identify the update delivery method.

Enumerator:

QAPI_NET_LWM2M_FOTA_UPDATE_PULL_E Supports only the package method.
QAPI_NET_LWM2M_FOTA_UPDATE_PUSH_E Supports only the package URI method.
QAPI_NET_LWM2M_FOTA_UPDATE_BOTH_E Supports both the package and package URI methods.

22.1.4.7 enum qapi_Net_LWM2M_Location_Resource_Id_t

Enum to identify the location resource ID.

Enumerator:

QAPI_NET_LWM2M_LOCATION_RES_O_RADIUS_E Location resource is the radius.

22.1.4.8 enum qapi_Net_LWM2M_SW_Mgnt_Resource_Id_t

Enum to identify a particular resource of a software management object.

Enumerator:

QAPI_NET_LWM2M_SW_MGNT_RES_O_PACKAGE_NAME_E Resource ID for Package Name.
QAPI_NET_LWM2M_SW_MGNT_RES_O_PACKAGE_VERSION_E Resource ID for Package Version.

QAPI_NET_LWM2M_SW_MGNT_RES_O_PACKAGE_E Resource ID for Package.
QAPI_NET_LWM2M_SW_MGNT_RES_O_PACKAGE_URI_E Resource ID for Package URI.
QAPI_NET_LWM2M_SW_MGNT_RES_M_INSTALL_E Resource ID for Install.
QAPI_NET_LWM2M_SW_MGNT_RES_M_UNINSTALL_E Resource ID for Uninstall.
QAPI_NET_LWM2M_SW_MGNT_RES_M_UPDATE_STATE_E Resource ID for Update State.
QAPI_NET_LWM2M_SW_MGNT_RES_M_UPDATE_RESULT_E Resource ID for Update Result.
QAPI_NET_LWM2M_SW_MGNT_RES_M_ACTIVATE_E Resource ID for Activate.
QAPI_NET_LWM2M_SW_MGNT_RES_M_DEACTIVATE_E Resource ID for Deactivate.
QAPI_NET_LWM2M_SW_MGNT_RES_M_ACTIVATION_STATE_E Resource ID for Activation State.

22.1.4.9 enum qapi_Net_LWM2M_SW_Mgnt_Error_Value_t

Enum to identify a particular error value of a software management object.

Enumerator:

QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_INITIAL_E Update result is Initial.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_DOWNLOADING_E Update result is Downloading.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_INSTALL_SUCCESS_E Update result is Install Success.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_NO_ENOUGH_STORAGE_E Update result is Not Enough Storage.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_OUT_OF_MEMORY_E Update result is Device is Out of Memory.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_CONNECTION_LOST_E Update result is Connection Lost.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_PKG_CHECK_FAILURE_E Update result is Package Check Failure.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_PKG_UNSUPPORTED_E Update result is Package Unsupported.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_INVALID_URI_E Update result is Invalid URI.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_UPDATE_ERROR_E Update result is Update Error.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_INSTALL_ERROR_E Update result is Install Error.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_RES_UNINSTALL_ERROR_E Update result is Uninstall Error.

22.1.4.10 enum qapi_Net_LWM2M_SW_Mgnt_State_t

Enum to identify the particular state of a software management object.

Enumerator:

QAPI_NET_LWM2M_SW_MGNT_UPDATE_STATE_INITIAL_E Update state is Initial.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_STATE_DOWNLOAD_STARTED_E Update state is Download Started.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_STATE_DOWNLOADED_E Update state is Downloaded.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_STATE_DELIVERED_E Update state is Delivered.
QAPI_NET_LWM2M_SW_MGNT_UPDATE_STATE_INSTALLED_E Update state is Installed.

22.1.4.11 enum qapi_Net_Firmware_State_t

Enum to identify the particular state of a firmware object.

Enumerator:

QAPI_NET_LWM2M_FIRWARE_STATE_IDLE_E Firmware state is Idle.
QAPI_NET_LWM2M_FIRWARE_STATE_DOWNLOADING_E Firmware state is Downloading.
QAPI_NET_LWM2M_FIRWARE_STATE_DOWNLOADED_E Firmware state is Downloaded.
QAPI_NET_LWM2M_FIRWARE_STATE_UPDATING_E Firmware state is Updating.

22.1.4.12 enum qapi_Net_LWM2M_ID_t

Enum to identify the type of ID set in the LWM2M object information.

Enumerator:

QAPI_NET_LWM2M_OBJECT_ID_E Object ID.
QAPI_NET_LWM2M_INSTANCE_ID_E Instance ID.
QAPI_NET_LWM2M_RESOURCE_ID_E Resource ID.
QAPI_NET_LWM2M_RESOURCE_INSTANCE_ID_E Resource instance ID.

22.1.4.13 enum qapi_Net_LWM2M_Value_Type_t

Enum to identify the type of resource value.

Enumerator:

QAPI_NET_LWM2M_TYPE_UNDEFINED Resource value type is Undefined.
QAPI_NET_LWM2M_TYPE_OBJECT Resource value type is Object.
QAPI_NET_LWM2M_TYPE_OBJECT_INSTANCE Resource value type is Object Instance.
QAPI_NET_LWM2M_TYPE_MULTIPLE_RESOURCE Resource value type is Multiple Resource.
QAPI_NET_LWM2M_TYPE_STRING_E Resource value type is String.
QAPI_NET_LWM2M_TYPE_OPAQUE_E Resource value type is Opaque.
QAPI_NET_LWM2M_TYPE_INTEGER_E Resource value type is Integer.
QAPI_NET_LWM2M_TYPE_FLOAT_E Resource value type is Float.
QAPI_NET_LWM2M_TYPE_BOOLEAN_E Resource value type is Boolean.
QAPI_NET_LWM2M_TYPE_OBJECT_LINK Resource value type is Object Link.

22.1.4.14 enum qapi_Net_LWM2M_Write_Attr_t

LWM2M write attribute types.

Enumerator:

QAPI_NET_LWM2M_MIN_PERIOD_E Minimum period.
QAPI_NET_LWM2M_MAX_PERIOD_E Maximum period.
QAPI_NET_LWM2M_GREATER_THAN_E Greater than.
QAPI_NET_LWM2M_LESS_THAN_E Less than.
QAPI_NET_LWM2M_STEP_E Step.
QAPI_NET_LWM2M_DIM_E Dimension.

22.1.4.15 enum qapi_Net_LWM2M_DL_Msg_t

LWM2M downlink message types.

Enumerator:

QAPI_NET_LWM2M_READ_REQ_E Read request.
QAPI_NET_LWM2M_WRITE_REPLACE_REQ_E Write replace request.
QAPI_NET_LWM2M_WRITE_PARTIAL_UPDATE_REQ_E Write partial update request.
QAPI_NET_LWM2M_WRITE_ATTR_REQ_E Write attribute request.
QAPI_NET_LWM2M_DISCOVER_REQ_E Discover request.
QAPI_NET_LWM2M_EXECUTE_REQ_E Execute request.
QAPI_NET_LWM2M_DELETE_REQ_E Delete request.
QAPI_NET_LWM2M_OBSERVE_REQ_E Observe request.
QAPI_NET_LWM2M_CANCEL_OBSERVE_REQ_E Cancel observe request.
QAPI_NET_LWM2M_ACK_MSG_E Acknowledge message.
QAPI_NET_LWM2M_INTERNAL_CLIENT_IND_E Internal client indication.
QAPI_NET_LWM2M_CREATE_REQ_E Create request.
QAPI_NET_LWM2M_DELETE_ALL_REQ_E Delete all request.

22.1.4.16 enum qapi_Net_LWM2M_UL_Msg_t

LWM2M uplink message types.

Enumerator:

QAPI_NET_LWM2M_RESPONSE_MSG_E Response message.
QAPI_NET_LWM2M_NOTIFY_MSG_E Notify message.
QAPI_NET_LWM2M_CREATE_RESPONSE_MSG_E Create response message.

22.1.4.17 enum qapi_Net_LWM2M_Event_t

LWM2M event information.

Enumerator:

QAPI_NET_LWM2M_STATE_INITIAL_E Initial state.
QAPI_NET_LWM2M_BOOTSTRAP_REQUIRED_E Bootstrap required event.
QAPI_NET_LWM2M_BOOTSTRAP_COMPLETED_E Bootstrap completed event.
QAPI_NET_LWM2M_BOOTSTRAP_FAILED_E Bootstrap failed event.
QAPI_NET_LWM2M_REGISTERTION_COMPELTED_E Registration completed event.
QAPI_NET_LWM2M_REGISTRATION_FAILED_E Registration failed event.
QAPI_NET_LWM2M_DEVICE_REBOOT_E Device reboot event.
QAPI_NET_LWM2M_DEVICE_FACTORY_RESET_E Device factory reset event.
QAPI_NET_LWM2M_DEVICE_REBOOTSTRAPPING_E Device rebootstrapping event.
QAPI_NET_LWM2M_TX_MESSAGE_MAX_RETRY_FAILURE_E Tx message maximum retry failure event.
QAPI_NET_LWM2M_RX_MESSAGE_INTERNAL_FAILURE_E Rx message internal failure event.
QAPI_NET_LWM2M_SLEEP_E Sleep event.
QAPI_NET_LWM2M_WAKEUP_E Wake-up event.
QAPI_NET_LWM2M_CLIENT_RESET_E Reset event.

QAPI_NET_LWM2M_LIFETIME_UPDATE_E Lifetime update event.
QAPI_NET_LWM2M_REGISTER_UPDATE_E Register update event.
QAPI_NET_LWM2M_BOOTSTRAP_START_E Bootstrap required event.

22.1.4.18 enum qapi_Net_LWM2M_Response_Code_t

LWM2M response status codes.

Enumerator:

QAPI_NET_LWM2M_IGNORE_E Ignore.
QAPI_NET_LWM2M_201_CREATED_E 201 - Created.
QAPI_NET_LWM2M_202_DELETED_E 202 - Deleted.
QAPI_NET_LWM2M_204_CHANGED_E 204 - Changed.
QAPI_NET_LWM2M_205_CONTENT_E 205 - Content.
QAPI_NET_LWM2M_400_BAD_REQUEST_E 400 - Bad request.
QAPI_NET_LWM2M_401_UNAUTHORIZED_E 401 - Unauthorized.
QAPI_NET_LWM2M_402_BAD_OPTION_E 402 - Bad option.
QAPI_NET_LWM2M_403_FORBIDDEN_E 403 - Forbidden.
QAPI_NET_LWM2M_404_NOT_FOUND_E 404 - Not found.
QAPI_NET_LWM2M_405_METHOD_NOT_ALLOWED_E 405 - Method is not allowed.
QAPI_NET_LWM2M_406_NOT_ACCEPTABLE_E 406 - Not acceptable.
QAPI_NET_LWM2M_408_ENTITY_INCOMPLETE_E 408 - Request Entity Incomplete.
QAPI_NET_LWM2M_413_ENTITY_TOO_LARGE_E 413 - Request entity too large.
QAPI_NET_LWM2M_415_UNSUPPORTED_DATA_FORMAT_E 415 - Unsupported content format.
QAPI_NET_LWM2M_500_INTERNAL_SERVER_E 500 - Internal server.

22.1.4.19 enum qapi_Net_LWM2M_Content_Type_t

LWM2M message content type.

Enumerator:

QAPI_NET_LWM2M_TEXT_PLAIN Plain text.
QAPI_NET_LWM2M_TEXT_XML XML text.
QAPI_NET_LWM2M_TEXT_CSV CSV text.
QAPI_NET_LWM2M_TEXT_HTML HTML text.
QAPI_NET_LWM2M_APPLICATION_LINK_FORMAT Application link format.
QAPI_NET_LWM2M_APPLICATION_XML Application XML.
QAPI_NET_LWM2M_APPLICATION_OCTET_STREAM Application Octet stream.
QAPI_NET_LWM2M_APPLICATION_RDF_XML Application RDF XML.
QAPI_NET_LWM2M_APPLICATION_SOAP_XML Application SOAP XML.
QAPI_NET_LWM2M_APPLICATION_ATOM_XML Application ATOM XML.
QAPI_NET_LWM2M_APPLICATION_XMPP_XML Application XMPP XML.
QAPI_NET_LWM2M_APPLICATION_EXI Application EXI.
QAPI_NET_LWM2M_APPLICATION_FASTINFOSET Application FastInfoSet.
QAPI_NET_LWM2M_APPLICATION_SOAP_FASTINFOSET Application SOAP FastInfoSet.
QAPI_NET_LWM2M_APPLICATION_JSON Application JSON.
QAPI_NET_LWM2M_APPLICATION_X_OBIX_BINARY Application X OBIX binary.
QAPI_NET_LWM2M_M2M_TLV M2M TLV.

QAPI_NET_LWM2M_M2M_JSON M2M JSON.

22.1.4.20 enum qapi_Net_LWM2M_Config_Type_t

LWM2M configuration parameter type.

Enumerator:

QAPI_NET_LWM2M_CONFIG_BOOTSTRAP_URL Configure the bootstrap URL.
QAPI_NET_LWM2M_CONFIG_APN_NAME Configure the APN name.
QAPI_NET_LWM2M_CONFIG_SECURITY_MODE Configure the security mode.

22.1.4.21 enum qapi_Net_LWM2M_Security_Mode_t

LWM2M security mode type.

Enumerator:

QAPI_NET_LWM2M_SECURITY_MODE_PRE_SHARED_KEY Preshared Key mode.
QAPI_NET_LWM2M_SECURITY_RAW_PUBLIC_KEY Raw Public Key mode.
QAPI_NET_LWM2M_SECURITY_CERTIFICATE Security Certificate mode.
QAPI_NET_LWM2M_SECURITY_NONE No security mode.

22.2 LWM2M APIs

This section provides the LWM2M APIs.

22.2.1 Typedef Documentation

22.2.1.1 typedef qapi_Status_t(∗ qapi_Net_LWM2M_App_CB_t)(qapi_Net_LWM2M_-App_Handler_t handle, qapi_Net_LWM2M_Data_t ∗lwm2m_data)

Callback registered from the application, which is used by the LWM2M client to indicate the resource value change to the application.

Parameters

in	*handle*	Handle received from qapi_Net_LWM2M_Register_App_Extended().
in	*lwm2m_data*	Pointer to the LWM2M data.

Returns

See Section 11.1.
On success, QAPI_OK(0) is returned. Other value on error.

22.2.1.2 typedef qapi_Status_t(∗ qapi_Net_LWM2M_App_Extended_CB_t)(qapi_-Net_LWM2M_App_Handler_t handle, qapi_Net_LWM2M_Server_Data_t ∗lwm2m_srv_data, void ∗user_data)

Callback registered from the application, which is used by the LWM2M client to indicate any extended object-specific messages from the server to the appropriate application. Each server message request is associated with a message ID and passed to the caller as part of the LWM2M server. The application must maintain the message ID to message mapping and use the message ID for any futher transactions that involve responses or notification events pertaining to the message.

Parameters

in	*handle*	Handle received from qapi_Net_LWM2M_Register_App_Extended().
in	*lwm2m_srv_data*	Pointer to the LWM2M server data.
in	*user_data*	Pointer to the user data.

Returns

See Section 11.1.
On success, QAPI_OK(0) is returned. Other value on error.

22.2.2 Function Documentation

22.2.2.1 qapi_Status_t qapi_Net_LWM2M_Register_App_Extended (qapi_Net_LWM2M_App_Handler_t * *handle*, void * *user_data*, qapi_Net_LWM2M_App_Extended_CB_t *user_cb_fn*)

Registers an application with an LWM2M client along with a callback handle. The application gets a handle on successful registration with the LWM2M client and must use this handle for subsequent calls to the LWM2M client in the APIs.

Parameters

in,out	*handle*	Handle that is provided to the application on successful registration.
in	*user_data*	Transparent user data payload (to be returned in the user callback).
in	*user_cb_fn*	User client callback handle to forward data to the application.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.2 qapi_Status_t qapi_Net_LWM2M_DeRegister_App (qapi_Net_LWM2M_App_Handler_t *handle*)

Deregisters an application. Any object instances associated with the handle are automatically cleaned up as a result of deregistration.

Parameters

in	*handle*	Handle that was provided to the application on successful registration.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.3 qapi_Status_t qapi_Net_LWM2M_Observe (qapi_Net_LWM2M_App_Handler_t *handle*, qapi_Net_LWM2M_App_CB_t *observe_cb_fn*, qapi_Net_LWM2M_Object_Info_t * *observe_info*)

Used by the application to indicate to the LWM2M client the object/instance/resource that the application is interested in observing. Only allowed for standard objects.

Parameters

in	*handle*	Handle received after successful application registration.
in	*observe_cb_fn*	Application callback to be invoked on a value change.
in	*observe_info*	Object/instance/resource information that the application is interested in monitoring on on the LWM2M client.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.4 qapi_Status_t qapi_Net_LWM2M_Cancel_Observe (qapi_Net_LWM2M_App_Handler_t *handle,* qapi_Net_LWM2M_Object_Info_t * *observe_info*)

Used by the application to cancel the observation.

Parameters

in	*handle*	Handle received after successful application registration.
in	*observe_info*	Object/instance/resource information for which the application is to cancel the observation.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.5 qapi_Status_t qapi_Net_LWM2M_Create_Object_Instance (qapi_Net_LWM2M_App_Handler_t *handle,* qapi_Net_LWM2M_Data_t * *lwm2m_data*)

Creates standard/custom LWM2M object instances from the application. Only one object instance is allowed at a time. Applications are allowed to create instances of standard objects at any time and can pass the information associated with the instance. However, custom/extensible object instances can only be created by the application within the bootstrap window during the bootstrap phase. If the application missed the bootstrap window internally, rebootstrapping can be set to force the device to perform rebootstrapping on the next reboot, and the application is then allowed to create the new object instance. It is not required by the application to pass the information of the custom object instance.

Parameters

in	*handle*	Handle received after successful application registration.
in	*lwm2m_data*	LWM2M object instance and its resource information.

Returns

See Section 11.1.

On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.6 qapi_Status_t qapi_Net_LWM2M_Delete_Object_Instance (qapi_Net_LWM2M_App_Handler_t *handle,* qapi_Net_LWM2M_Object_Info_t * *instance_info*)

Deletes an LWM2M object instance from the application. Only one object instance deletion is allowed at a time.

Parameters

in	*handle*	Handle received after successful application registration.
in	*instance_info*	LWM2M object instance and its resource information.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.7 qapi_Status_t qapi_Net_LWM2M_Get (qapi_Net_LWM2M_App_Handler_t *handle,* qapi_Net_LWM2M_Object_Info_t * *lwm2m_info_req,* qapi_Net_LWM2M_Data_t ** *lwm2m_data*)

Gets the value of the LWM2M object/instance/resource from the application. Only one query of an object instance is allowed at a time.

Parameters

in	*handle*	Handle received after successful application registration.
in	*lwm2m_info_req*	Object/instance/resource information requested from the application.
out	*lwm2m_data*	Value of the LWM2M object/instance/resource information.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.8 qapi_Status_t qapi_Net_LWM2M_Set (qapi_Net_LWM2M_App_Handler_t *handle,* qapi_Net_LWM2M_Data_t * *lwm2m_data*)

Sets the value of LWM2M resources. Only one object instance setting is allowed at a time.

Note that only the following resources are available to be set (per the OMA Specificaion):

- Firmware update (by kernel applications only)
  - (3) State
  - (5) Update Result

  - (6) PkgName
  - (7) PkgVersion
  - (8) Firmware Update Protocol Support
  - (9) Firmware Update Delivery Method
- Software management object
  - (7) Update State
  - (9) Update Result
  - (12) Activation State
- Device capability
  - (0) Property
  - (1) Group
  - (2) Description
  - (3) Attached
  - (4) Enabled
- Device object
  - (0) Manufacturer
  - (1) Model Number
  - (2) Serial Number (by kernel applications only)
  - (3) Firmware Version (by kernel applications only)
  - (18) Hardware Version (by kernel applications only)
  - (19) Software Version (by kernel applications only)

Parameters

in	*handle*	Handle received after successful application registration.
in	*lwm2m_data*	Value of the LWM2M resource to be set.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.9 qapi_Status_t qapi_Net_LWM2M_Send_Message (qapi_Net_LWM2M_App_Handler_t *handle,* qapi_Net_LWM2M_App_Ex_Obj_Data_t * *lwm2m_app_data*)

Sends application data, which can be responses or notification events, to the server. For notifications, a notification ID is returned by the LWM2M client, and it is the application's responsiblity to store this notification ID. If there is an observation cancellation, the LWM2M client will send this notification ID through the registered callback. Applications can encode the data payload using their own encode functions.

Parameters

in	*handle*	Handle received after successful application registration.
in,out	*lwm2m_app_data*	Value of the LWM2M extended/custom object information to be sent. The application is responsible for releasing any allocated resources.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.10 qapi_Status_t qapi_Net_LWM2M_Encode_Data (qapi_Net_LWM2M_App_Handler_t *handle*, qapi_Net_LWM2M_Obj_Info_t * *obj_info*, qapi_Net_LWM2M_Flat_Data_t * *in_dec_data*, size_t *in_dec_data_size*, qapi_Net_LWM2M_Attributes_t * *write_attr*, qapi_Net_LWM2M_Content_Type_t *enc_content_type*, uint8_t ** *out_enc_data*, uint32_t * *out_enc_data_len*)

Utility function to encode application response/notification data before sending them to the server. If applications have their own encoding functions, they are free to use those functions to encode the data payload.

Parameters

in	*handle*	Handle received after successful application registration.
in	*obj_info*	Object/URI information.
in	*in_dec_data*	Input data that is to be encoded.
in	*in_dec_data_size*	Input data size (in buffers).
in	*write_attr*	Write attribute information.
in	*enc_content_type*	Encoding format of the data.
out	*out_enc_data*	Output data buffer in encoded format. Resources are allocated internally. The application is responsible for releasing any allocated resources.
out	*out_enc_data_len*	Output encoded data buffer length.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.11 qapi_Status_t qapi_Net_LWM2M_Decode_Data (qapi_Net_LWM2M_App_-Handler_t *handle*, qapi_Net_LWM2M_Obj_Info_t * *obj_info*, uint8_t * *in_enc_data*, uint32_t *in_enc_data_len*, qapi_Net_LWM2M_Content_Type_t *dec_content_type*, qapi_Net_LWM2M_Flat_Data_t ** *out_dec_data*, size_t * *out_dec_data_size*)

Utility function to decode the server request data received through the registered application callback. If applications have their own decoding functions, they are free to use those functions to decode the data payload.

Parameters

in	*handle*	Handle received after successful application registration.
in	*obj_info*	Object/URI information.
in	*in_enc_data*	Input data that is to be decoded.
in	*in_enc_data_len*	Input data length.
in	*dec_content_type*	Decoding format of the input data.
out	*out_dec_data*	Output data buffer in decoded format. Resources are allocated internally. The application is responsible for releasing any allocated resources.
out	*out_dec_data_size*	Output decoded data size (in buffers).

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.12 qapi_Status_t qapi_Net_LWM2M_Wakeup (qapi_Net_LWM2M_App_-Handler_t *handle*, uint8_t * *msg_id*, uint8_t *msg_id_len*)

Wakes up the LWM2M client module to send notifications to the server.

Wake-up and Sleep states of the LWM2M client are indicated to the application using the qapi_net_LWM2M_Server_Data_t.event registered callback. The application is responsible for tracking the states of the LWM2M client.

Parameters

in	*handle*	Handle received after successful application registration.
in	*msg_id*	Message ID information associated with the request.
in	*msg_id_len*	Message ID information length.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.13 qapi_Status_t qapi_Net_LWM2M_Default_Attribute_Info (qapi_Net_LWM2M_App_Handler_t *handle*, uint16_t *server_id*, uint32_t * *p_min*, uint32_t * *p_max*)

Gets the value of the default Pmin and Pmax information for a specific server.

Parameters

in	*handle*	Handle received after successful application registration.
in	*server_id*	Server ID information (use QAPI_LWM2M_SERVER_ID_INFO macro).
out	*p_min*	Default "p_min" server attribute value.
out	*p_max*	Default "p_max" server attribute value.

Returns

See Section 11.1.
On success, QAPI_OK(0) is returned. Other value on error.

22.2.2.14 qapi_Status_t qapi_Net_LWM2M_Set_ServerLifeTime (qapi_Net_LWM2M_App_Handler_t *handle*, uint8_t * *url_info*, uint32_t *life_time*)

Configures the server life time information in the LWM2M client from the application. If the device is connected to only a single server, it is optional to pass the URL information.

Parameters

in	*handle*	Handle received after successful application registration.
in	*url_info*	URL information of the server.
in	*life_time*	Server life time information to be configured.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

22.2.2.15 qapi_Status_t qapi_Net_LWM2M_Get_ServerLifeTime (qapi_Net_LWM2M_App_Handler_t *handle*, uint8_t * *url_info*, uint32_t * *life_time*)

Retrieves the server life time information from the LWM2M client to the application. If the device is connected to only a single server, it is optional to pass the URL information.

Parameters

in	*handle*	Handle received after successful application registration.
in	*url_info*	URL information of the server.

out	*life_time*	Server life time information that is configured.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

23 AT Forward Service Framework

23.1 AT Forward Macros

23.1.1 Define Documentation

23.1.1.1 #define qapi_atfwd_Pass_Pool_Ptr(*a, b*) atfwd_set_byte_pool(a,b)

Macro that passes a Byte Pool pointer for the ATFWD application.

Parameter a – Pointer to the callback corresponding with the AT commands registered by the client with ATFWD.

Parameter b – Pointer to the Byte Pool.

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

Note: This macro is only used in the DAM space.

23.1.1.2 #define qapi_atfwd_release_byte_pool(*a*) atfwd_release_byte_pool(a)

Macro that releases a Byte Pool pointer for the ATFWD application.

Parameter a – Pointer to the callback corresponding with the AT commands registered by the client with ATFWD.

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

Note: This macro is only used in the DAM space.

23.2 Register New AT Commands

23.2.1 Function Documentation

23.2.1.1 qapi_Status_t qapi_atfwd_reg (char * *name,* at_fwd_cb_type *atfwd_callback*)

Registers new custom AT commands along with their respective callbacks.

Parameters

in	*name*	Pointer to an AT commands string.
in	*atfwd_callback*	Pointer to the callback corresponding to the AT commands.

Returns

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

23.3 Deregister an AT Command

23.3.1 Function Documentation

23.3.1.1 qapi_Status_t qapi_atfwd_dereg (char * *name*)

Deregisters AT commands.

Parameters

in	*name*	Pointer to the AT commands list.

Returns

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

23.4 Send a Response

23.4.1 Function Documentation

23.4.1.1 qapi_Status_t qapi_atfwd_send_resp (char * *atcmd_name,* char * *buf,* uint32_t *result*)

Sends a response.

Parameters

in	*atcmd_name*	Pointer to the particular AT command to which this response corresponds.
in	*buf*	Pointer to the buffer containing the response.
in	*result*	0 – Result ERROR. This is to be set in case of ERROR or CME ERROR. The response buffer contains the entire details. 1 – Result OK. This is to be set if the final response is to send an OK result code to the terminal.

Returns

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

23.5 Send a URC Response

23.5.1 Function Documentation

23.5.1.1 qapi_Status_t qapi_atfwd_send_urc_resp (char * *atcmd_name,* char * *at_urc*)

Sends a URC response.

Parameters

in	*atcmd_name*	Pointer to the particular AT command to which this response corresponds.
in	*at_urc*	Pointer to the buffer containing the response.

Returns

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

24 QAPI Utility APIs

This chapter contains APIs for QAPI utility functions.

- Driver Access APIs for the DAM Application Space
- Command Line Interface

24.1 Driver Access APIs for the DAM Application Space

This section provides helper function that can access a list to store user space pointer details for DAM applications.

24.1.1 Function Documentation

24.1.1.1 qapi_Status_t qapi_data_map_u_addr_to_handle (void * *handle,* unsigned int *app_type,* void * *u_addr,* unsigned int *addr_len*)

Utility API to map a user space address to the handle. Only needs to be used in the DAM Application space.

Parameters

in	*handle*	Networking stack module handle.
in	*app_type*	Identifies the networking stack module.
in	*u_addr*	User space address pointer.
in	*addr_len*	Size of the user space address that is allocated.

Returns

QAPI_OK on success or <0 on failure.

24.1.1.2 qapi_Status_t qapi_data_map_handle_to_u_addr (void * *handle,* unsigned int *app_type,* void ** *u_addr*)

Utility API to retrieve the user space address corresponding to the handle. Only needs to be used in the DAM Application space.

Parameters

in	*handle*	Networking stack module handle.
in	*app_type*	Identifies the networking stack module.
out	*u_addr*	User space address pointer mapped to the handle.

Returns

QAPI_OK on success or <0 on failure.

24.2 Command Line Interface

24.2.1 Data Structure Documentation

24.2.1.1 struct qapi_CLI_Parameter_t

This structure contains the information for a single parameter entered into the command line.

Data fields

Type	Parameter	Description
uint8_t *	String_Value	String value.
int32_t	Integer_Value	Integer value (if the string was successfully converted).
bool	Integer_Is_-Valid	Flag indicating whether the integer is valid.

24.2.1.2 struct qapi_CLI_Command_t

Structure that represents the information for a single command in a command list.

Data fields

Type	Parameter	Description
qapi_CLI_-Command_-Function_t	Command_-Function	Function to be called when the command is executed from the CLI.
bool	Start_Thread	Flag that indicates whether the command is to start on its own thread.
const uint8_t *	Command_-String	String representation of the function.
const uint8_t *	Usage_String	Usage string for the command.
const uint8_t *	Description	Description string for the commmand.
qapi_CLI_-Command_-Function_a	abort_function	Abort function.

24.2.1.3 struct qapi_CLI_Command_Group_t

Structure that represents a command group that can be registered with the CLI.

Data fields

Type	Parameter	Description
const uint8_t *	group_string	String representation of the group.
uint32_t	command_-count	Number of commands in the group.
const qapi_CL-I_Command_t *	command_list	List of commands for the group.

24.2.1.4 struct qapi_CLI_Parameter_Data_t

Structure that represents command parameters.

Data fields

Type	Parameter	Description
uint8_t	index	Index of the command group corresponding to the command.
uint32_t	parameter_-count	Number of parameters passed in the command.
qapi_CLI_-Parameter_t *	parameter_list	List of parameters.

24.2.2 Typedef Documentation

24.2.2.1 typedef qapi_Status_t(* qapi_CLI_Command_Function_t)(uint32_t parameter_count, qapi_CLI_Parameter_t *parameter_list)

Type that represents the format of a function that processes commands from the CLI.

Parameters

handle	Handle received from qapi_CLI_Register_Command_Group().
parameter_count	Number of parameters that were specified to the CLI for the function.
parameter_list	List of parameters specified to the CLI for the function.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

24.2.3 Function Documentation

24.2.3.1 qapi_Status_t qapi_CLI_Register_Command_Group (qapi_CLI_App_-Handler_t * *handle*, void * *user_data*, qapi_CLI_Command_Group_t * *command_group*, qapi_CLI_Command_CB_t *user_cb_fn*)

Registers a command group with the CLI.

Parameters

in,out	*handle*	Handle that is provided to the application on successful registration.
in	*user_data*	Transparent user data payload (to be returned in the user callback).
in	*command_group*	Command group to be registered. Note that this function assumes the command group information will be constant and simply stores a pointer to the data. If the command group and its associated information is not constant, its memory must be retained until the command is unregistered.
in	*user_cb_fn*	User client callback handle to forward data to the application.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

24.2.3.2 qapi_Status_t qapi_CLI_Unregister_Command_Group (qapi_CLI_App_-Handler_t * *handle*)

Unregisters a command group from the CLI.

Parameters

in	*handle*	Handle for the group to be unregistered. This will be the value returned form qapi_CLI_Register_Command_Group() when the function was registered.

Returns

See Section 11.1.
On success, QAPI_OK (0) is returned. Other value on error.

A TLS/DTLS Supported Ciphersuites

The ciphersuites in the following table are supported for transport layer security (TLS) and datagram transport layer security (DTLS).

Ciphersuite	Defined ciphersuite's name	TLS1.2/ DTLS1.2 supported ciphers	TLS1.1, TLS1.0, or DTLS 1.0 supported ciphers only
	TLS_NULL_WITH_NULL_NULL	No	No
PSK (preshared keys)	TLS_PSK_WITH_RC4_128_SHA	No	No
	TLS_PSK_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_PSK_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_PSK_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_PSK_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_PSK_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_PSK_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_PSK_WITH_AES_256_CBC_SHA384	Yes	No
ECDHE_ECDSA (Ephemeral Elliptic Curve Diffie-Hellman with Elliptic Curve Digital Signature Algorithm key)	TLS_ECDHE_ECDSA_WITH_NULL_SHA	No	No
	TLS_ECDHE_ECDSA_WITH_RC4_128_SHA	No	No
	TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	Yes	No
ECDH_ECDSA (Elliptic Curve Diffie-Hellman with Elliptic Curve Digital Signature Algorithm key)	TLS_ECDH_ECDSA_WITH_NULL_SHA	No	No
	TLS_ECDH_ECDSA_WITH_RC4_128_SHA	No	No
	TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDH_ECDSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDH_ECDSA_WITH_AES_256_GCM_SHA384	Yes	No

Ciphersuite	Defined ciphersuite's name	TLS1.2/ DTLS1.2 supported ciphers	TLS1.1, TLS1.0, or DTLS 1.0 supported ciphers only
ECDHE_RSA	TLS_ECDHE_RSA_WITH_NULL_SHA	No	No
	TLS_ECDHE_RSA_WITH_RC4_128_SHA	No	No
	TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_ SHA256	Yes	No
ECDH_RSA	TLS_ECDH_RSA_WITH_NULL_SHA	No	No
	TLS_ECDH_RSA_WITH_RC4_128_SHA	No	No
	TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_ECDH_RSA_WITH_AES_256_CBC_SHA384	Yes	No
	TLS_ECDH_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_ECDH_RSA_WITH_AES_256_GCM_SHA384	Yes	No
DHE_RSA (Diffie-Hellman signed using RSA keys)	TLS_DHE_RSA_WITH_DES_CBC_SHA	No	No
	TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA	No	No
	TLS_DHE_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	Yes	No
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	Yes	No
	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_DHE_RSA_WITH_AES_128_CCM	Yes	No
	TLS_DHE_RSA_WITH_AES_256_CCM	Yes	No
	TLS_DHE_RSA_WITH_AES_128_CCM_8	Yes	No
	TLS_DHE_RSA_WITH_AES_256_CCM_8	Yes	No
	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_ SHA256	Yes	No
RSA	TLS_RSA_WITH_NULL_MD5	No	No
	TLS_RSA_WITH_NULL_SHA	No	No
	TLS_RSA_WITH_RC4_128_MD5	No	No
	TLS_RSA_WITH_RC4_128_SHA	No	No
	TLS_RSA_WITH_DES_CBC_SHA	Yes	Yes
	TLS_RSA_WITH_3DES_EDE_CBC_SHA	No	Yes
	TLS_RSA_WITH_AES_128_CBC_SHA	Yes	Yes
	TLS_RSA_WITH_AES_256_CBC_SHA	Yes	Yes
	TLS_RSA_WITH_NULL_SHA256	No	No
	TLS_RSA_WITH_AES_128_CBC_SHA256	Yes	No

Ciphersuite	Defined ciphersuite's name	TLS1.2/ DTLS1.2 supported ciphers	TLS1.1, TLS1.0, or DTLS 1.0 supported ciphers only
	TLS_RSA_WITH_AES_256_CBC_SHA256	Yes	No
	TLS_RSA_WITH_AES_128_GCM_SHA256	Yes	No
	TLS_RSA_WITH_AES_256_GCM_SHA384	Yes	No
	TLS_RSA_WITH_AES_128_CCM	Yes	No
	TLS_RSA_WITH_AES_256_CCM	Yes	No
	TLS_RSA_WITH_AES_128_CCM_8	Yes	No
	TLS_RSA_WITH_AES_256_CCM_8	Yes	No

B References

B.1 Related Documents

Title	Number
Qualcomm Technologies	
MDM9206 Data Features Overview	80-P8101-7
MDM9206 Lightweight M2M User Guide	80-P8101-15
QAPI Overview for Application Developers	80-P8101-26
MDM9206 ThreadX QAPI Usage Guide	80-P8101-35

B.2 Acronyms and Terms

Acronym or term	Definition
APN	Access point name
BSD	Berkeley Software Distribution
CA	Certificate authority
CE	Call end
CLI	Command line interface
CSR	Certificate signing request
DHCP	Dynamic Host Configuration Protocol
DNS	Domain name or system
DSS	Data services sockets
DTLS	Datagram transport layer security
MTU	Maximum transmission unit
netctrl	Net control
PDP	Packet Data Protocol
PSK	Preshared key
QAPI	Qualcomm API
QMI	Qualcomm messaging interface
SPI	Serial peripheral interface
SSL	Secure sockets layer
TLS	Transport layer security
URC	Unsolicited result code