Qualcomm
TECHNOLOGIES, INC

QMI Common Service Interface API

Interface Specification

80-N1123-2 C

August 5, 2014

QUALCOMM®
2016-05-16 01:39:46 PDT
dawn_zhang@askey.com.tw

Confidential and Proprietary – Qualcomm Technologies, Inc.

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

Restricted Distribution: Not to be distributed to anyone who is not an employee of either Qualcomm or its subsidiaries without the express approval of Qualcomm's Configuration Management.

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

Qualcomm reserves the right to make changes to the product(s) or information contained herein without notice. No liability is assumed for any damages arising directly or indirectly by their use or application. The information provided in this document is provided on an "as is" basis.

This document contains confidential and proprietary information and must be shredded when discarded.

Qualcomm and QuRT are trademarks of QUALCOMM Incorporated, registered in the United States and other countries. All QUALCOMM Incorporated trademarks are used with permission. Other product and brand names may be trademarks or registered trademarks of their respective owners.

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

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

© 2011, 2014 Qualcomm Technologies, Inc.
All rights reserved.

Contents

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

QUALCOMM®
2016-05-25 11:29:05 IST
deon_zhang@sky.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Figures

Tables

QUALCOMM®
2016-05-16 01:39:46 PDT
deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Revision History

Revision	Date	Description
A	Feb 2011	Initial release.
B	Dec 2011	Updated document title. Added Section 3.2.2. Updated Sections 3.1.3, 3.3.1, 3.3.2, and Section 3.3.3.
C	Aug 2014	Numerous changes were made to this document. It should be read in its entirety.

QUALCOMM
2016-05-16 01:39:46 PDT
deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

1 Introduction

NOTE: Numerous changes were made to this document. It should be read in its entirety.

1.1 Purpose

This document explains the Qualcomm Messaging Interface (QMI) Common Service Interface (QCSI) API functions. These functions can be used in conjunction with the autogenerated files from the QMI IDL compiler to write a service that can receive and respond to messages from a client, as well as send indication messages. This document reflects the latest functions and behaviors of the QCSI framework.

1.2 Scope

This document is for customers who are familiar with the QMI and want to develop a service that runs on a modem processor.

The API functions in this document are subject to change based on further discussion within the team. A major change is, however, not expected.

1.3 Conventions

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

Parameter types are indicated by arrows:

- → Designates an input parameter
- ← Designates an output parameter
- ↔ Designates a parameter used for both input and output

Shading indicates content that has been added or changed in this revision of the document.

QUALCOMM®
2016-05-16 01:19:46 PDT
deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

1.4 References

Reference documents are listed in Table 1-1. Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm Technologies		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*QMI Client API Reference Guide*	80-N1123-1

1.5 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.

1.6 Acronyms

For definitions of terms and abbreviations, refer to [Q1]. Table 1-2 lists terms that are specific to this document.

Table 1-2 Acronyms

Acronym	Definition
QCSI	QMI common service interface
QMI	Qualcomm messaging interface
TCB	task control block

QUALCOMM 2016-05-10 01:39:46 PDT deon_zhang@asus.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

2 Theory of Operation

The QCSI simplifies the process of writing a QMI service by providing a set of functions and callbacks for handling requests, sending responses and indications, and handling client connect and disconnect events. The QCSI also encodes and decodes messages, so routines for encoding/decoding do not need to be written for each new message introduced to a service.

2.1 QCSI Handles

A service registers with the QCSI by calling qmi_csi_register, which takes in several callbacks and handles that are used through the rest of the service's interactions with the QCSI. The callbacks and handles are defined in Chapter 3. The following sections provide more details about the handles and how they are used in each function of the service's interaction with the QCSI. There are two basic types of handles: those provided by a service and those provided by the QCSI framework.

2.1.1 Handles Provided by Services

2.1.1.1 service_cookie

When registering with the QCSI framework, the service provides a service_cookie that contains server-specific context information that will be passed back in the qmi_csi_connect, qmi_csi_disconnect, and qmi_csi_process_req callbacks. This information can be anything that a service could find useful (e.g., a structure or a counter), but it can also be NULL.

2.1.1.2 connection_handle

A service provides the connection_handle to the framework as a return value from the qmi_csi_connect callback. This handle is used to identify the connection between a service and the client that has connected, and is passed back to the service in the qmi_csi_process_req and qmi_csi_disconnect callbacks, so the service can identify which client is sending a request or disconnecting.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

2.1.2 Handles Provided by QCSI Framework

2.1.2.1 service_provider

The service_provider handle is provided to the service from the QCSI framework as a return value from the qmi_csi_register function. This handle is used by the framework to identify the service when the service calls the qmi_csi_handle_event, qmi_csi_send_broadcast_ind, and qmi_csi_unregister functions.

2.1.2.2 client_handle

The client_handle is given to the service by the QCSI framework in the qmi_csi_connect callback. The handle allows the framework to identify the client for which the indication is intended when the service calls the qmi_csi_send_ind function.

2.1.2.3 req_handle

The req_handle is provided to the service by the QCSI framework in the qmi_csi_process_req callback. The framework uses this handle to match a response with its request when the service calls the qmi_csi_send_resp function.

2.2 Handling Events with QCSI

The QCSI framework does not have its own task and is designed for all service events to be handled in the service task context, which prevents blocking the QCSI from handling other services. This is accomplished with the signaling provided by the os_params variable and the qmi_csi_handle_event function.

2.2.1 os_params

When the QCSI receives an event for a specific service, i.e., client connect, disconnect, or request message, the QCSI framework uses the signaling information that was provided in the os_params value that was passed in the qmi_csi_register function. For details, see Chapter 6.

QUALCOMM®
2015-07-20 01:39:46 PDT
deon_zhang@sky.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

2.2.2 Processing in the Service's Context

When the service is signaled, the service calls qmi_csi_handle_event, which then calls the appropriate callback to handle the specific event. This takes place within the service's task, thereby preventing services from blocking each other.

The following pseudocode shows a basic handler loop for this behavior, which is illustrated in Section 2.3.

```
while (1)
{
   /* Wait on the appropriate signal/event in the os_params */
   /* Once the wait is unblocked, clear the signal/event and continue */
   /* qmi_csi_handle_event will call the appropriate callbacks
      within the service task context */
   rc = qmi_csi_handle_event(service_handle, os_params);
   if (rc != QMI_CSI_NO_ERR)
   {
      /* Do some error handling */
   }
        /* All message and even processing will take place in callbacks,
           once this point is reached, all processing should be complete,
           and it is safe to return to the top of the loop and wait on the
           signal/event again. */
{
```

QUALCOMM® 2016-05-16 11:19:46 PDT deon_zhang@asky.com

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

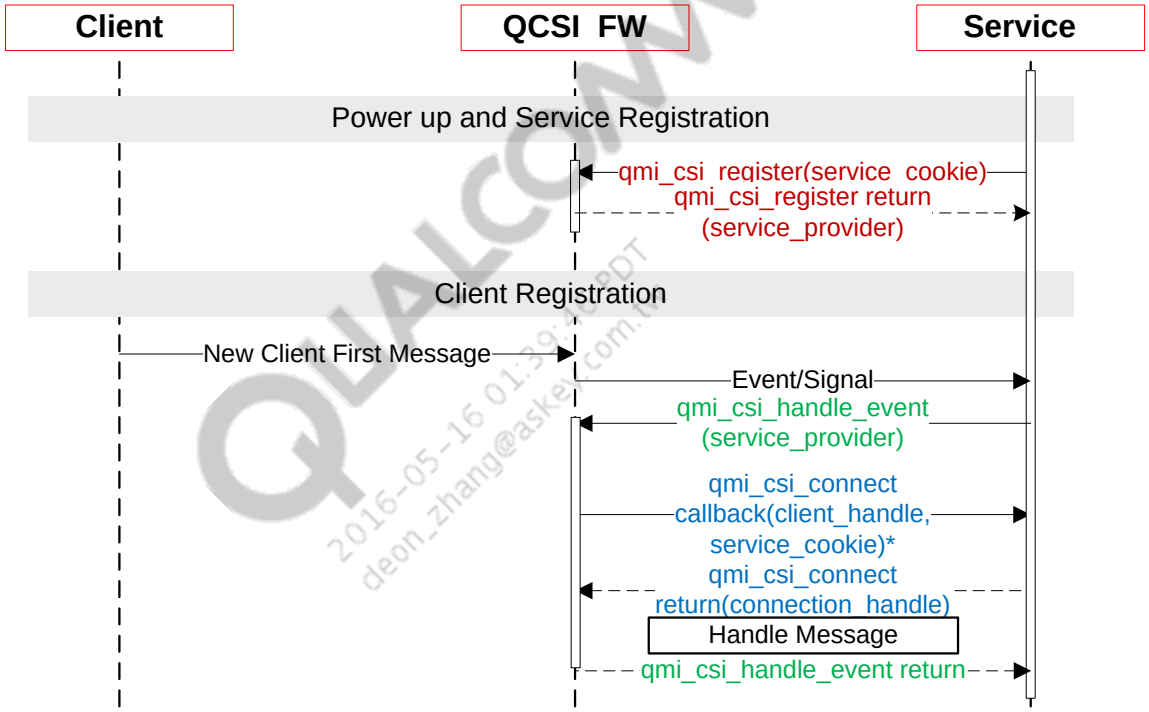
2.3 Call Flows

The following diagrams show the basic call flows for different service events.

NOTE: Function calls and their returns are color coded, and the handles that are provided in the function calls as input or output parameters are indicated in parentheses.

2.3.1 Initialization/registration

Figure 2-1 is the QCSI service initialization call flow.



*Client Connection is only received by QCSI when the client sends their first message.

Figure 2-1 QCSI service initialization

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

2.3.2 Synchronous Response

Figure 2-2 is the QCSI service synchronous response call flow.

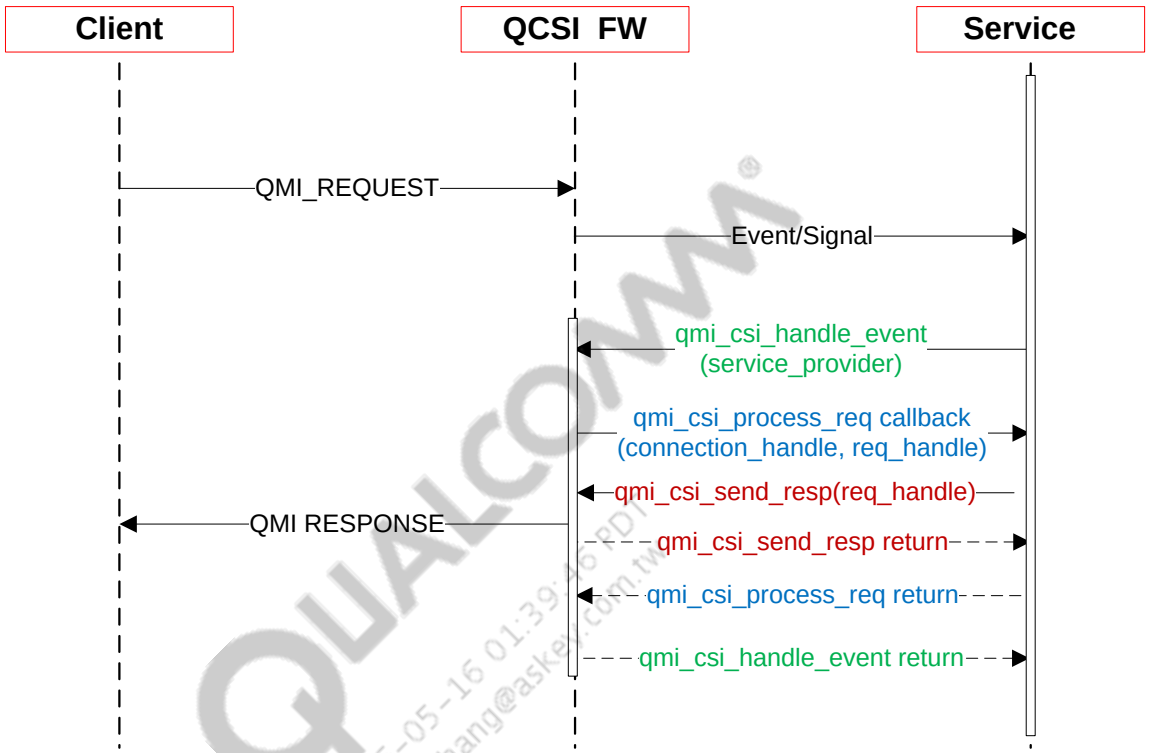


Figure 2-2 QCSI service synchronous response

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

2.3.3 Asynchronous Response

Figure 2-3 is the QCSI service asynchronous response call flow.

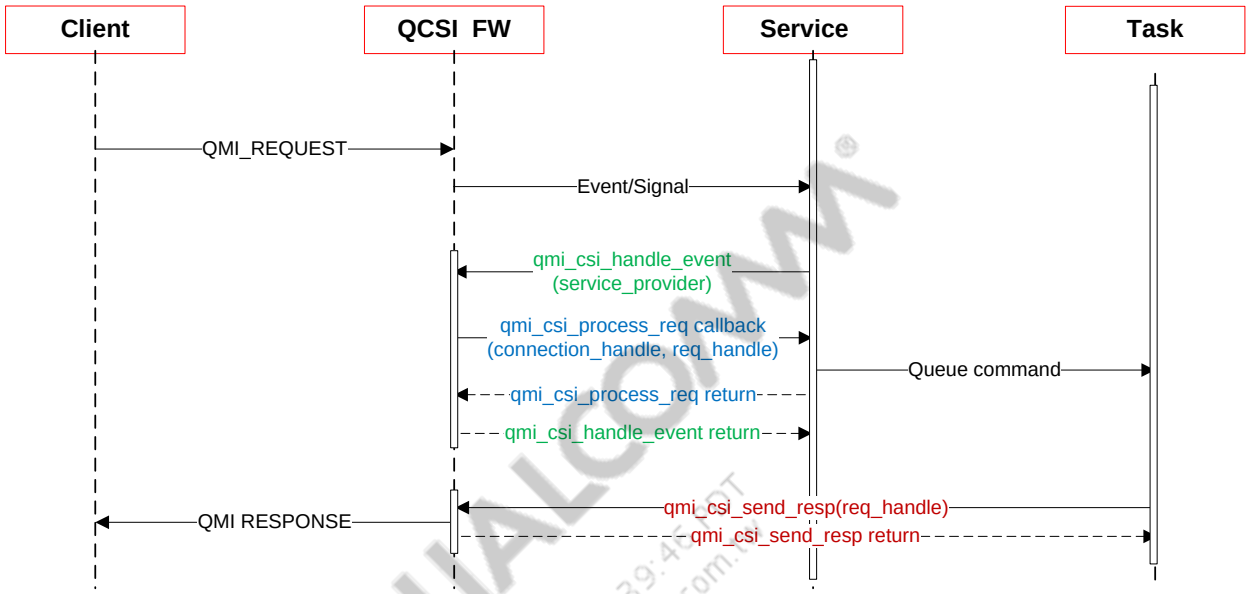


Figure 2-3 QCSI service asynchronous response

2.3.4 Indications

Figure 2-4 is the QCSI service indications call flow.

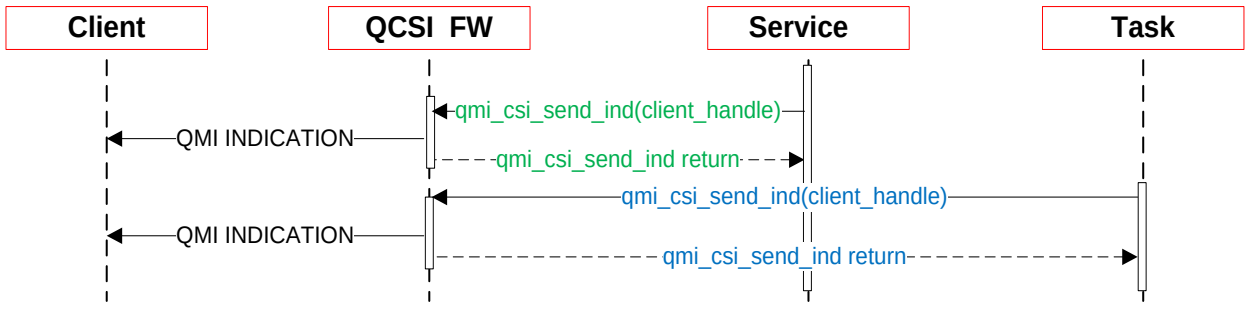


Figure 2-4 QCSI service indications

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3 QCSI API

The QCSI functions can be divided into the following four broad categories that are defined in the qmi_csi.h header file:

- Callback function prototypes
- Registration functions
- Message sending functions
- Event handling functions

3.1 Callback Function Prototypes

3.1.1 qmi_csi_connect

The QCSI framework calls this callback function when it receives the first request from each client.

Parameters

```
qmi_csi_cb_error
qmi_csi_connect
(
        qmi_client_handle       client_handle,
        void                    *service_cookie,
        void                    **connection_handle
);
```

→	client_handle	The framework uses this handle to identify the client that is connecting
→	service_cookie	Service-specific data that was provided as a parameter to qmi_csi_register
←	connection_handle	Services return this handle as a token to represent this client connection

Returns

QMI_CSI_CB_NO_ERR – If successful.

ERROR code – If unsuccessful.

Dependencies

None.

QUALCOMM 2016-05-16 07:39:46 PDT deon_zhang@asket.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3.1.2 qmi_csi_disconnect

This callback function is called by the QCSI framework when a client disconnects.

Parameters

```
void
qmi_csi_disconnect
(
  void                                *connection_handle,
  void                                *service_cookie,
);
```

→	connection_handle	Handle provided by the service in qmi_csi_connect
→	service_cookie	Service-specific data that was provided by the service in qmi_csi_register

Returns

None.

Dependencies

None.

3.1.3 qmi_csi_process_req

The QCSI framework calls this callback function after a message is received and the service calls the qmi_csi_handle_event function. The framework decodes the data and gives it to the service.

Parameters

```
qmi_csi_cb_error
qmi_csi_process_req
(
  void                                *connection_handle,
  qmi_req_handle                      req_handle,
  unsigned int                        msg_id,
  void                                *req_c_struct,
  unsigned int                        req_c_struct_len,
  void                                *service_cookie
);
```

→	connection_handle	Service provides this handle in qmi_csi_connect
→	req_handle	Framework provides this handle to identify this specific transaction and message
→	msg_id	Message ID for this specific message
→	req_c_struct	C structure with the decoded message

QUALCOMM® 2015-05-16 01:39:46 PDT dedi.zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

→	req_c_struct_len	C data structure size
→	service_cookie	Service-specific data that was provided by the service in qmi_csi_register

Returns

QMI_CSI_CB_NO_ERR – If successful.

ERROR code – If unsuccessful.

Dependencies

None.

3.2 Registration Functions

3.2.1 qmi_csi_register

Registers a service with the QCSI framework.

Parameters

```
qmi_csi_error
qmi_csi_register
(
  qmi_idl_service_object_type   service_obj,
  qmi_csi_connect               service_connect,
  qmi_csi_disconnect            service_disconnect,
  qmi_csi_process_req            service_process_req,
  void                          *service_cookie,
  qmi_csi_os_params             *os_params,
  qmi_csi_service_handle        *service_provider
);
```

→	service_obj	Object containing meta-information to encode and decode messages
→	service_connect	Callback that handles new client connections
→	service_disconnect	Callback that handles client disconnects
→	service_process_req	Callback that handles the incoming requests
→	service_cookie	Service-specific context that is passed to the connect, disconnect, and process request callbacks
→	os_params	OS-specific parameter used for signaling (see Chapter 6)
←	service_provider	Framework provides this handle to represent this service connection

QUALCOMM 2016-05-16 01:39:46 PDT deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Returns

QMI_CSI_NO_ERR – If successful.

ERROR code – If unsuccessful.

Dependencies

None.

3.2.2 qmi_csi_register_with_options

Registers a service with options with the QCSI framework. See Chapter 4 for more information about the available options.

Parameters

```
qmi_csi_error
qmi_csi_register_with_options
(
  qmi_idl_service_object_type    service_obj,
  qmi_csi_connect                service_connect,
  qmi_csi_disconnect             service_disconnect,
  qmi_csi_process_req             service_process_req,
  void                           *service_cookie,
  qmi_csi_os_params              *os_params,
  qmi_csi_options                *options,
  qmi_csi_service_handle         *service_provider
);
```

→	service_obj	Object containing meta-information to encode and decode messages
→	service_connect	Callback that handles new client connections
→	service_disconnect	Callback that handles client disconnects
→	service_process_req	Callback that handles incoming requests
→	service_cookie	Service-specific context that is passed to the connect, disconnect, and process request callbacks
→	os_params	OS-specific parameter used for signaling (see Chapter 6)
→	options	Options defined by qmi_csi_options (see Chapter 4)
←	service_provider	Framework provides this handle to represent this service connection

Returns

QMI_CSI_NO_ERR – If successful.

ERROR code – If unsuccessful.

Dependencies

None.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3.2.3 qmi_csi_unregister

Unregisters a server. This function must never be called in the connect, disconnect, or handle request callbacks.

Parameters

```
qmi_csi_error
qmi_csi_unregister
(
  qmi_csi_service_handle              service_provider
);
```

→	service_provider	Handle provided by the framework in qmi_csi_register

Returns

QMI_CSI_NO_ERR – If successful.

ERROR code – If unsuccessful.

Dependencies

None.

3.3 Message Sending Functions

3.3.1 qmi_csi_send_resp

Sends a response to the client.

Parameters

```
qmi_csi_error
qmi_csi_send_resp
(
  qmi_req_handle                      req_handle,
  unsigned int                        msg_id,
  void                                *resp_c_struct,
  unsigned int                        resp_c_struct_len
);
```

→	req_handle	Handle provided by the framework in the process request callback
→	msg_id	Message ID for this specific message
→	resp_c_struct	C data structure for the response
→	resp_c_struct_len	C data structure size

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Returns

QMI_CSI_NO_ERR – Sets the transaction handle on success

Error code – If unsuccessful.

Dependencies

None.

3.3.2 qmi_csi_send_ind

Sends an indication to the client.

Parameters

```
qmi_csi_error
qmi_csi_send_ind
(
  qmi_client_handle              client_handle,
  unsigned int                   msg_id,
  void                           *ind_c_struct,
  unsigned int                   ind_c_struct_len
);
```

→	client_handle	Handle provided by the framework in the connect callback
→	msg_id	Message ID for this specific message
→	ind_c_struct	C data structure for this indication
→	ind_c_struct_len	C data structure size

Returns

QMI_CSI_NO_ERR – Sets the transaction handle on success

Error code – If unsuccessful.

Dependencies

None.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3.3.3 qmi_csi_send_broadcast_ind

Sends a broadcast indication to all registered clients.

Parameters

```
qmi_csi_error
qmi_csi_send_broadcast_ind
(
  qmi_csi_service_handle   service_provider,
  unsigned int             msg_id,
  void                     *ind_c_struct,
  unsigned int             ind_c_struct_len
);
```

→	service_provider	Handle provided by the framework in qmi_csi_register
→	msg_id	Message ID for this specific indication
→	ind_c_struct	C data structure for this broadcast indication
→	ind_c_struct_len	C data structure size

Returns

QMI_CSI_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

QUALCOMM® 2016-05-16 01:39:46 PDT deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3.4 Event Handling Functions

3.4.1 qmi_csi_handle_event

Called to handle an event after the server thread receives an event notification. Callbacks from qmi_csi_register will be invoked in the server's context.

Parameters

```
qmi_csi_error
qmi_csi_handle_event
(
  qmi_csi_server_handle   service_provider,
  qmi_csi_os_params       *os_params
);
```

→	service_provider	Handle provided by the framework in qmi_csi_register
→	os_params	OS-specific parameter used for signaling (see Chapter 6)

Returns

QMI_CSI_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

QUALCOMM 2016-05-16 01:39:46 PM deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4 QCSI Options

4.1 Options Use in QCSI

The addition of the qmi_csi_register_with_options function allows services to modify the default behavior of their service; the values in the options parameter are changed in the service. Helper macros exist to manage all of the options in the structure before registration is performed.

4.1.1 QMI_CSI_OPTIONS_INIT

Initializes all elements of the options structure to their defaults.

Parameters

```
QMI_CSI_OPTIONS_INIT
(
  qmi_csi_options  options
);
```

→	options	Options structure to be initialized.

Returns

QMI_CSI_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

QUALCOMM®
2015-05-16 01:39:46 PDT
deon.zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.1.2 QMI_CSI_OPTIONS_SET_INSTANCE_ID

Allows a service to specify a unique instance ID, which allows multiple instances of the same service to exist and be identified in the system.

Parameters

```
QMI_CSI_OPTIONS_SET_INSTANCE_ID
(
  qmi_csi_options  options,
  unsigned int     instance
);
```

→	options	Options structure to be initialized.
→	instance	Instance ID to assign to this service.

Returns

QMI_CSI_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

4.1.3 QMI_CSI_OPTIONS_SET_SCOPE

Sets the security scope for the service. This is used if you want your service to be visible *only* on the processor on which it registers (not broadcast to any other processors).

Parameters

```
QMI_CSI_OPTIONS_SET_SCOPE
(
  qmi_csi_options  options,
  uint64_t         scope
);
```

→	options	Options structure to be initialized.
→	scope	Supports local only scope.

QUALCOMM 2015-05-16 01:39:46 PDT Ren_Zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Returns

QMI_CSI_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

4.1.4 QMI_CSI_OPTIONS_SET_MAX_OUTSTANDING_INDS

Sets the maximum number of indications that are allowed to be in flight (outstanding).

Parameters

```
QMI_CSI_OPTIONS_SET_MAX_OUTSTANDING_INDS
(
  qmi_csi_options  options,
  unsigned int     max_inds
);
```

→	options	Options structure to be initialized.
→	max_inds	Number of indications to buffer before QMI_CSI_CONN_BUSY is returned.

Returns

QMI_CSI_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.1.5 QMI_CSI_OPTIONS_SET_RAW_REQUEST_CB

Sets a raw request message handler. Any messages received by the service that are not defined in the IDL are passed in as raw (still encoded) and the service can decode them manually.

Parameters

```
QMI_CSI_OPTIONS_SET_RAW_REQUEST_CB
(
  qmi_csi_options     options,
  qmi_csi_process_req raw_req_cb
);
```

→	options	Options structure to be initialized.
→	raw_req_cb	Request callback to handle raw messages.

Returns

QMI_CSI_CB_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

4.1.6 QMI_CSI_OPTIONS_SET_PRE_REQUEST_CB

Sets a pre-request handler callback, which the QCSI framework calls before anything is done with the incoming request message.

Parameters

```
QMI_CSI_OPTIONS_SET_PRE_REQUEST_CB
(
  qmi_csi_options     options,
  qmi_csi_process_req pre_req_cb
);
```

→	options	Options structure to be initialized.
→	pre_req_cb	Request callback to handle messages prior to framework management.

QUALCOMM® 2011-05-16 01:39:46 PDT shun_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Detailed description

The service can then decide the next task to perform:

- Handle this message raw (returns QMI_CSI_CB_REQ_HANDLED)
- Request the framework to go ahead and decode the message and call the request callback (returns QMI_CSI_CB_NO_ERR)
- Refuse the request message (returns an error code other than QMI_CSI_CB_NO_ERR or QMI_CSI_CB_REQ_HANDLED)

Returns

QMI_CSI_CB_REQ_HANDLED or QMI_CSI_CB_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

4.1.7 QMI_CSI_OPTIONS_SET_RESUME_IND_CB

Sets a Tx resume handler that the framework calls when a previously busy client is now available to accept indications. This function is only called at some point after a call to qmi_csi_send_ind() returns QMI_CSI_CONN_BUSY.

Parameters

```
QMI_CSI_OPTIONS_SET_RESUME_IND_CB
(
  qmi_csi_options     options,
  qmi_csi_resume_ind  resume_cb
);
```

→	options	Options structure to be initialized.
→	resume_cb	Callback called when an indication queue has been emptied.

Returns

QMI_CSI_CB_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.1.8 QMI_CSI_OPTIONS_SET_REQ_HANDLER_TBL

Sets a request handler callback table, which the QCSI framework uses to call specific handlers for specific message IDs. This allows a service to define which message IDs it supports on a per-target basis, and allows different messages to be decoded by the framework or passed in raw.

Parameters

```
QMI_CSI_OPTIONS_SET_REQ_HANDLER_TBL
(
  qmi_csi_options               options,
  qmi_csi_req_handler_tbl_type  req_handler_tbl,
  uint16_t                      req_handler_size
);
```

→	options	Options structure to be initialized.
→	req_handler_tbl	Table of request handler callback information.
→	req_handler_size	Number of entries in request handler table.

Detailed description

The qmi_csi_req_handler_tbl_type_s structure is:

```
typedef struct qmi_csi_req_handler_tbl_type_s
  {
    uint16_t            msg_id;
    uint8_t             decoded;
    qmi_csi_process_req  service_process_req;
  }qmi_csi_req_handler_tbl_type;
```

The req_handler_tbl is an array of the qmi_csi_req_handler_tbl_type entries, where:

- msg_id is the message to be handled
- decoded is a boolean value that allows a service to specify whether or not the message ID is to be decoded by the framework
- service_process_req is a function pointer to a handler function

Returns

QMI_CSI_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4.1.9 QMI_CSI_OPTIONS_SET_LOG_MSG_CB

Sets a logging message callback that the QCSI framework calls when QMI messages are sent or received.

Parameters

```
QMI_CSI_OPTIONS_SET_LOG_MSG_CB
(
  qmi_csi_options     options,
  qmi_csi_log_msg     log_msg_cb
);
```

→	options	Options structure to be initialized.
→	log_msg_cb	Callback called when a message is sent or received.

Returns

QMI_CSI_CB_NO_ERR – If successful.

Error code – If unsuccessful.

Dependencies

None.

QUALCOMM®
2016-05-16 01:39:46 PDT
deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

5 Error Codes

5.1 QMI_CSI_ERROR

Table 5-1 lists the error values in the qmi_csi_error enum.

Table 5-1 QMI CSI error values

Error code
QMI_CSI_NO_ERR
QMI_CSI_CONN_REFUSED
QMI_CSI_CONN_BUSY
QMI_CSI_INVALID_HANDLE
QMI_CSI_INVALID_ARGS
QMI_CSI_ENCODE_ERR
QMI_CSI_DECODE_ERR
QMI_CSI_NO_MEM
QMI_CSI_INTERNAL_ERR

5.2 QMI_CSI_CB_ERROR

Table 5-2 lists the error values in the qmi_csi_cb_error enum.

Table 5-2 QMI CSI CB error values

Error code
QMI_CSI_CB_NO_ERR
QMI_CSI_CB_CONN_REFUSED
QMI_CSI_CB_NO_MEM
QMI_CSI_CB_INTERNAL_ERR
QMI_CSI_CB_UNSUPPORTED_ERR
QMI_CSI_CB_REQ_HANDLED

QUALCOMM® 2016-05-16 01:39:46 EDT jon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

6 OS-specific Parameters

Each OS defines a unique qmi_csi_os_params structure to hold the required OS-specific signaling information.

6.1 Rex (Modem)

The qmi_csi_os_params structure in rex is defined as follows:

```
typedef struct
{
  rex_tcb_type  *tcb;
  rex_sigs_type  sig;
} qmi_csi_os_params;
```

The service provides the Task Control Block (TCB) of the service worker thread and the signal mask that the framework is to set when it requires the service to handle events for the service. The service is expected to wait in the service worker thread and handle events. An example of the expected worker loop is provided below:

```
qmi_csi_service_handle handle;
qmi_csi_os_params os_params;

os_params.tcb = rex_self();
os_params.sig = 0x1;

/* Register with service using qmi_csi_register()
 * and get handle */

while(1)
{
  rex_wait(0x1);
  rex_clr_sigs(rex_self(), 0x1);
  qmi_csi_handle_event(handle, &os_params);
}
```

NOTE: The service is free to handle multiple events from the same thread's work loop as long as it ensures that the signal 0x1 (in the case of the example) is reserved for the specific service.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

6.2 QuRT (ADSP)

The qmi_csi_os_params structure in QuRT™ is defined as follows:

```
typedef struct
{
  qurt_anysignal_t *signal;
  unsigned int     sig;
} qmi_csi_os_params;
```

The service provides the initialized signaling object it is going to wait on and the signal mask it is expecting to use for events from the QCSI framework. The service is expected to wait in the service worker thread and handle events. An example of the expected worker loop is provided below:

```
qmi_csi_service_handle handle;
qmi_csi_os_params os_params;
qurt_anysignal_t signal;

qurt_anysignal_init(&signal);
os_params.signal = &signal;
os_params.sig = 0x1;

/* Register with service using qmi_csi_register()
 * and get handle */

while(1)
{
  qurt_anysignal_wait(os_params.signal, 0x1);
  qurt_anysignal_clear(os_params.signal, 0x1);
  qmi_csi_handle_event(sp, &os_params);
}
```

NOTE: The service is free to handle multiple events on the same signaling object in the worker thread's work loop as long as it ensures that the signal 0x1 (in the case of the example) is reserved for the specific service.

QUALCOMM® 2015-05-16 01:39:46 PDT deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

6.3 Linux (Android™, Linux Enablement)

Services running in Linux® use the select() system call to wait for events on the specific service. The qmi_csi_os_params structure is defined as follows:

```
typedef struct
{
  fd_set fds;
  int    max_fd;
} qmi_csi_os_params;
```

The call to qmi_csi_register() or qmi_csi_register_with_options() treats the OS Parameters as an output parameter where the framework sets the file descriptors it has created for the specific service in the fds member and adjusts max_fd appropriately. The service code must initialize these values appropriately before the first call to qmi_csi_register*():

```
FD_ZERO(&os_params.fds);
os_params.max_fd = 0;
```

Once initialization is complete, the service is expected to wait in the service worker thread and handle events. An example of the expected worker loop is provided below. Note that qmi_csi_handle_event() inspects the fds member to check if it has events to handle:

```
qmi_csi_os_params os_params,os_params_in;
fd_set fds;
qmi_csi_service_handle handle;

/* Initialize os params before first register */
FD_ZERO(&os_params.fds);
os_params.max_fd = 0;

/* Register with service using qmi_csi_register()
 * and get handle */

while(1)
{
  fds = os_params.fds;
  FD_SET(STDIN_FILENO, &fds);
  select(os_params.max_fd+1, &fds, NULL, NULL, NULL);
  /* Test for user input of the ctrl+d character
   * to terminate the server */
```


MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

```
  if(FD_ISSET(STDIN_FILENO, &fds))
  {
    if(read(STDIN_FILENO, buf, sizeof(buf)) <= 0)
    {
      break;
    }
  }
  os_params_in.fds = fds;
  qmi_csi_handle_event(sp, &os_params_in);
}
```

NOTE: The service is free to handle multiple events on the same work loop and all it has to do is reuse the same os_params for each call to qmi_csi_register*().

6.4 QNX

6.4.1 Within the Privileged Process ('mis')

Within the privileged process, the pthread native signaling operations are used:

```
typedef struct {
    volatile uint32_t sig_set;
    pthread_cond_t cond;
    pthread_mutex_t mutex;
} qmi_csi_thread_signal_type;

typedef union {
  qmi_csi_thread_signal_type *thread_signal;
  qmi_csi_fd_signal_type     fd_signal;
} qmi_csi_os_params;
```

The user must provide a pointer to the initialized qmi_csi_thread_signal_type in the thread_signal pointer (the service code is expected to initialize the mutex, cond, and sig_set variables). An example service work loop is provided below:

```
qmi_csi_os_params os_params;
qmi_csi_service_handle handle;

thread_signal.sig_set = 0;
pthread_mutex_init(&thread_signal.mutex, NULL);
pthread_cond_init(&thread_signal.cond, NULL);
os_params.thread_signal = &thread_signal;
```


MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

```
/* Register with service using qmi_csi_register()
 * and get handle */

while(1)
{
  pthread_mutex_lock(&thread_signal.mutex);
  if(!thread_signal.sig_set) {
    pthread_cond_wait(&thread_signal.cond, &thread_signal.mutex);
  }
  thread_signal.sig_set = 0;
  pthread_mutex_unlock(&thread_signal.mutex);
  qmi_csi_handle_event(handle, &os_params);
}
```

The service can reuse the same os_params for multiple services if it wants to wait on multiple services using the same work loop.

6.4.2 Outside the Privileged Process

Services running outside the privileged process use the select() system call to wait for events on the specific service. The qmi_csi_os_params structure is defined as follows:

```
typedef struct {
    fd_set data_fds;
    fd_set ctrl_fds;
    int    max_fds;
}qmi_csi_fd_signal_type;

typedef union {
  qmi_csi_thread_signal_type *thread_signal;
  qmi_csi_fd_signal_type     fd_signal;
} qmi_csi_os_params;
```

The call to qmi_csi_register() or qmi_csi_register_with_options() treats the OS Parameters as an output parameter where the framework sets the file descriptors it has created for the specific service in the data_fds and ctrl_fds members and adjusts max_fds appropriately. The service code must initialize these values appropriately before the first call to qmi_csi_register*():

```
FD_ZERO(&os_params.data_fds);
FD_ZERO(&os_params.ctrl_fds);
os_params.max_fds = 0;
```


MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

Once initialization is complete, the service is expected to wait in the service worker thread and handle events. An example of the expected worker loop is provided below. Note that qmi_csi_handle_event() inspects the data_fds and ctrl_fds members to check if it has events to handle:

```
qmi_csi_os_params os_params,os_params_in;
fd_set data_fds, ctrl_fds;
qmi_csi_service_handle handle;

/* Initialize os params before first register */
FD_ZERO(&os_params.data_fds);
FD_ZERO(&os_params.ctrl_fds);
os_params.max_fda = 0;

/* Register with service using qmi_csi_register()
 * and get handle */

while(1)
{
  data_fds = os_params.data_fds;
  ctrl_fds = os_params.ctrl_fds;

  FD_SET(STDIN_FILENO, &data_fds);
  select(os_params.max_fds+1, &data_fds, NULL, &ctrl_fds, NULL);
  /* Test for user input of the ctrl+d character
   * to terminate the server */
  if(FD_ISSET(STDIN_FILENO, &data_fds))
  {
    if(read(STDIN_FILENO, buf, sizeof(buf)) <= 0)
    {
      break;
    }
  }
  os_params_in.data_fds = data_fds;
  os_params_in.ctrl_fds = ctrl_fds;
  qmi_csi_handle_event(handle, &os_params_in);
}
```

NOTE: The service is free to handle multiple events on the same work loop and all it has to do is reuse the same os_params for each call to qmi_csi_register*().

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

6.5 Windows

Services running in Windows® use the native windows signaling event objects, which change based on the location of the service (Kernel or User mode).

6.5.1 Windows Kernel Mode

```
typedef struct
{
  PRKEVENT event;
} qmi_csi_os_params;
```

The event of the os_params is to be initialized with a call to KeInitializeEvent(...) and it is to be defined as a NotificationEvent Type, with the initial state set to FALSE (unsignaled.)

Then the service is to wait in an event loop, waiting on the event and processing commands as they occur.

```
while(1)
{
  KeClearEvent(os_params.event);
  KeWaitForSingleObject(os_params.event, Executive, KernelMode,
                        FALSE,  NULL);
  qmi_csi_handle_event(handle, &os_params);
}
```

6.5.2 Windows User Mode

```
typedef struct
{
  HANDLE  rxPeekHandle;
  HANDLE  eventPeekHandle;
} qmi_csi_os_params;
```

Both events of the os_params must be initialized by the service with a call to CreateEvent. Then service is to wait in an event loop on both events, calling qmi_csi_handle_event whenever either event has been signaled.

```
while(1)
{
  HANDLE wait_handles[2];
  wait_handles[0] = os_params.rxPeekHandle;
  wait_handles[1] = os_params.eventPeekHandle;
  ResetEvent(wait_handles[0]);
```


MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

```
          ResetEvent(wait_handles[1]);
          WaitForMultipleObjectsEx(2, wait_handles, FALSE, INFINITE, FALSE);
          qmi_csi_handle_event(handle, &os_params);
        }
```

QUALCOMM®
2016-05-16 01:39:46 PDT
deon_zhang@askey.com.tw

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION