ACKNOWLEDGEMENT

By utilizing this website and/or documentation, I hereby acknowledge as follows:

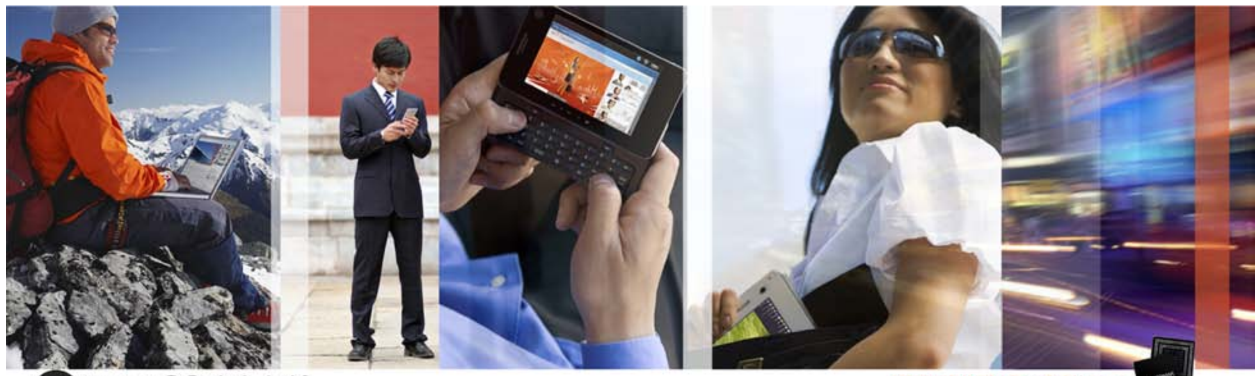
Effective October 1, 2012, QUALCOMM Incorporated completed a corporate reorganization in which the assets of certain of its businesses and groups, as well as the stock of certain of its direct and indirect subsidiaries, were contributed to Qualcomm Technologies, Inc. (QTI), a wholly-owned subsidiary of QUALCOMM Incorporated that was created for purposes of the reorganization.

Qualcomm Technology Licensing (QTL), the Company's patent licensing business, continues to be operated by QUALCOMM Incorporated, which continues to own the vast majority of the Company's patent portfolio. Substantially all of the Company's products and services businesses, including QCT, as well as substantially all of the Company's engineering, research and development functions, are now operated by QTI and its direct and indirect subsidiaries[1]. Neither QTI nor any of its subsidiaries has any right, power or authority to grant any licenses or other rights under or to any patents owned by QUALCOMM Incorporated.

No use of this website and/or documentation, including but not limited to the downloading of any software, programs, manuals or other materials of any kind or nature whatsoever, and no purchase or use of any products or services, grants any licenses or other rights, of any kind or nature whatsoever, under or to any patents owned by QUALCOMM Incorporated or any of its subsidiaries. A separate patent license or other similar patent-related agreement from QUALCOMM Incorporated is needed to make, have made, use, sell, import and dispose of any products or services that would infringe any patent owned by QUALCOMM Incorporated in the absence of the grant by QUALCOMM Incorporated of a patent license or other applicable rights under such patent.

Any copyright notice referencing QUALCOMM Incorporated, Qualcomm Incorporated, QUALCOMM Inc., Qualcomm Inc., Qualcomm or similar designation, and which is associated with any of the products or services businesses or the engineering, research or development groups which are now operated by QTI and its direct and indirect subsidiaries, should properly reference, and shall be read to reference, QTI.

[1] The products and services businesses, and the engineering, research and development groups, which are now operated by QTI and its subsidiaries include, but are not limited to, QCT, Qualcomm Mobile & Computing (QMC), Qualcomm Atheros (QCA), Qualcomm Internet Services (QIS), Qualcomm Government Technologies (QGOV), Corporate Research & Development, Qualcomm Corporate Engineering Services (QCES), Office of the Chief Technology Officer (OCTO), Office of the Chief Scientist (OCS), Corporate Technical Advisory Group, Global Market Development (GMD), Global Business Operations (GBO), Qualcomm Ventures, Qualcomm Life (QLife), Quest, Qualcomm Labs (QLabs), Snaptracs/QCS, Firethorn, Qualcomm MEMS Technologies (QMT), Pixtronix, Qualcomm Innovation Center (QuIC), Qualcomm iSkoot, Qualcomm Poole and Xiam.



QMI Core Server Framework APIs

Reference Guide

80-N7262-1 A

September 6, 2011

Submit technical questions at:
https://support.cdmatech.com/

QUALCOMM Incorporated
5775 Morehouse Drive
San Diego, CA 92121-1714
U.S.A.

Contents

Tables

Revision history

Revision	Date	Description
A	Sep 2011	Initial release

1 Introduction

1.1 Purpose

This document explains the Core Server Framework APIs. These APIs are built on top of the QCSI framework with the primary purpose of facilitating the clients to implement a server with less effort. The Framework allows the users to write an object-based server that extends the core server object. The core server object provides generic functionality that every service needs.

1.2 Scope

This document is for customers who are familiar with the Qualcomm Messaging Interface (QMI) and who wish to develop a service that runs on a modem processor.

The APIs mentioned in the document are subject to change based on further discussion within the team. However, a major change is 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

1.4 References

Reference documents, which may include QUALCOMM®, standards, and resource 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		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*QMI Common Service API Reference Guide*	80-N1123-2 A

1.5 Technical assistance

For assistance or clarification on information in this guide, submit a case to Qualcomm CDMA Technologies at https://support.cdmatech.com/.

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

1.6 Acronyms

For definitions of terms and abbreviations, see [Q1].

2 Core Server Framework APIs

The Core Server Framework APIs can be divided into six broad categories:

- Callback Function Prototypes
- Constructor/Destructor APIs
- Registration APIs
- Message Dispatch APIs
- Event Handling APIs
- Utility APIs

These are all defined in the header file qmi_core_server.h.

2.1 Callback function prototypes

2.1.1 qmi_csi_connect

This callback function is called by the QCSI framework when it receives a request from each client (user of a service).

Prototype

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

Parameters

→	client_handle	Handle used by the framework 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

2.1.2 qmi_csi_disconnect

This callback function is called by the QCSI framework when each client (user of a service) disconnects.

Prototype

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

Parameters

→	connection_handle	Handle provided by the service in qmi_csi_connect for the client disconnecting
→	service_cookie	Service-specific data that was provided as a parameter to qmi_csi_register

2.1.3 qmi_csi_process_req

This callback function is called by the QCSI framework 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.

Prototype

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

Parameters

→	connection_handle	Handle provided by the service in qmi_csi_connect
→	req_handle	Handle provided by the framework to identify this particular transaction and message
→	msg_id	Message ID pertaining to this particular message
→	req_c_struct	C structure with the decoded message
→	req_c_struct_len	Size of the C data structure
→	service_cookie	Service specific data that was provided as a parameter to qmi_csi_register

2.2 Constructor/Destructor APIs

2.2.1 qmi_core_server_new

This function constructs a new core server object.

Prototype

```
qmi_core_server_error_type
qmi_core_server_new
(
 qmi_core_server_object_type      *core_object,
 char                             *name,
 uint32_t                         instance_id,
 uint8_t                          task_flag,
 void                             *entry_func,
 void                             *priority,
 void                             *stk_size,
 void                             *sig,
 qmi_csi_os_params                *os_params,
 qmi_msg_handler_type             *disp_table,
 uint32_t                         table_size
);
```

Parameters

←	`core_object`	Pointer to a newly constructed core server object
→	`name`	Name of the core object
→	`instance_id`	Instance identifier associated with a server
→	`task_flag`	Task is created for the server if this flag is set, otherwise the server is assumed to be taskless
→	`entry_func`	Entry point for the task created by the framework; this argument should be NULL if task_flag is not set
→	`priority`	Priority used to create the task; this argument should be NULL if task_flag is not set
→	`stk_size`	The stack size used to create the task; this argument should be NULL if task_flag is not set.
→	`sig`	Signal used by the task created to wait on server events, should be NULL if task_flag is not set
→	`os_params`	Pointer to the signaling information for the taskless server; this parameter should be set to NULL if task_flag is set
→	`disp_table`	Pointer to dispatch function table, used to handle messages that the server understands
→	`table_size`	Size of the dispatch table

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.2.2 qmi_core_server_delete

This function destroys the core server object.

Prototype

```
qmi_core_server_error_type
qmi_core_server_delete
(
  qmi_core_server_object_type                *core_object
);
```

Parameters

→	`core_object`	Pointer to core object that has to be destroyed

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.3 Registration APIs

2.3.1 qmi_core_server_register

This function registers a server object with the QCSI infrastructure.

Prototype

```
qmi_core_server_error_type
qmi_core_server_register
(
  void                                        *server_obj,
  qmi_idl_service_object_type                  service_obj,
  qmi_csi_connect                              service_connect,
  qmi_csi_disconnect                           service_disconnect,
  qmi_csi_process_req                          service_process_req
);
```

Parameters

→	server_obj	Pointer to the server object
→	service_obj	Object containing meta information to encode/decode the messages
→	service_connect	Callback to register each client with the service
→	service_disconnect	Callback to unregister each client from service
→	service_process_req	Callback that handles the incoming requests

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.3.2 qmi_core_server_unregister

This function deregisters a server object with the QCSI infrastucture.

Prototye

```
qmi_core_server_error_type
qmi_core_server_unregister
(
  void           *server_obj
);
```

Parameters

→	`server_obj`	Pointer to the server object

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.3.3 qmi_core_server_register_client

This function registers a client with the framework and provides a connection object to represent the client. This function should be used from inside the service_connect callback.

Prototye

```
qmi_core_server_error_type
qmi_core_server_register_client
(
  qmi_core_conn_obj_type              *client_conn,
  qmi_core_server_object_type         *core_object,
  qmi_client_handle                   client_handle,
  uint32_t                            num_ind,
  void                                *client_data
);
```

Parameters

←	`client_conn`	Handle to represent a client connection
→	`core_object`	Pointer to core server object
→	`client_handle`	Handle used by QCSI infrastructure to identify the destination client
→	`num_ind`	Number of Indications implemented by the server
→	`client_data`	Client-specific data

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.3.4 qmi_core_server_unregister_client

This function unregisters a client from the framework. This function should be used from inside the service_disconnect callback. Also, this function should be called before the client deallocates the client data.

Prototye

```
qmi_core_server_error_type
qmi_core_server_unregister_client
(
  qmi_core_conn_obj_type              *client_conn
);
```

Parameters

→	`client_conn`	Handle to represent a client connection

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.4 Message Dispatch APIs

2.4.1 qmi_core_server_dispatch_msg

This function calls the appropriate handler of the incoming message.

Prototype

```
qmi_core_server_error_type
qmi_core_server_dispatch_msg
(
  qmi_core_conn_obj_type                        *client_conn,
  void                                          *server_obj,
  qmi_req_handle                                req_handle,
  int32_t                                       msg_id,
  void                                          *req_c_struct,
  int32_t                                       req_c_struct_len
);
```

Parameters

→	client_conn	Handle to represent a client connection
→	server_obj	Pointer-to-server object that extends the core server object
→	req_handle	Handle provided by the infrastructure to specify this particular transaction and message
→	msg_id	Message ID for this particular message
→	req_c_struct	C struct with the decoded data
→	req_c_struct_len	Length of c struct

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.4.2 qmi_core_server_send_ind

This function sends an indication to the client.

Prototype

```
 qmi_core_server_error_type
 qmi_core_server_send_ind
 (
    qmi_core_server_object_type               *core_object,
    int32_t                                   msg_id,
    void                                      *res_c_struct,
    int32_t                                   res_c_struct_len
);
```

Parameters

→	core_object	Pointer to core server object
→	msg_id	Message ID for this particular message
→	res_c_struct	C data structure for this indication
→	res_c_struct_len	Size of the C data structure

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.4.3 qmi_core_server_send_resp

This function sends a response to a client.

Prototype

```
qmi_core_server_error_type
qmi_core_server_send_resp
(
  qmi_req_handle               req_handle,
  int32_t                      msg_id,
  void                         *c_struct,
  int32_t                      c_struct_len
);
```

Parameters

→	`req_handle`	Handle used by the infrastructure to identify the transaction and the destination client
→	`msg_id`	Message ID for this particular message
→	`c_struct`	C data structure for this response
→	`c_struct_len`	Size of the response c struct

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.5 Event Handling APIs

2.5.1 qmi_core_server_handle_event

This function is called to handle an event after the server thread receives an event notification. Callbacks from qmi_core_server_register will be invoked in the server's context.

Prototype

```
qmi_core_server_error_type
qmi_core_server_handle_event
(
  void              *server_obj
);
```

Parameters

→	`server_obj`	Pointer to server object

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.6 Utility APIs

2.6.1 qmi_core_server_start_server

This function starts the server thread. It must not be used if the server is taskless.

Prototype

```
qmi_core_server_error_type
qmi_core_server_start_server
(
  void                             *server_obj
);
```

Parameters

→	server_obj	Pointer to server object

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.6.2 qmi_core_server_check_valid_object

This function checks if the core server object passed in the core server framework APIs was created by the constructor qmi_core_server_new.

Prototype

```
qmi_core_server_error_type
qmi_core_server_check_valid_object
(
   void                                    *server_obj
);
```

Parameters

→	server_obj	Pointer to server object

Return value

This function returns:

- QMI_CORE_SERVER_NO_ERR – Successful
- ERROR code – Unsuccessful

2.6.3 qmi_core_server_get_client_data

This function retrieves the client-specific data that was passed while registering a client.

Prototype

```
void*
qmi_core_server_get_client_data
(
   qmi_core_conn_obj_type                      *conn_obj
);
```

Parameters

→	conn_obj	Pointer to connection object

Return value

This function returns:

- Client-specific data – Successful
- NULL – Unsuccessful

2.6.4 qmi_core_server_get_os_params

This function retrieves the signaling information in case the server is created with a task.

Prototype

```
qmi_csi_os_params*
qmi_core_server_get_os_params
(
   qmi_core_server_object_type                    *core_object
);
```

Parameters

→	`core_object`	Pointer to core object

Return value

This function returns:

- OS params – Successful
- NULL – Unsuccessful

A Error Codes

A.1 QMI_CORE_SERVER_ERROR_TYPE

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

Table A-1 QMI CSI ERROR values

Error code
QMI_CORE_SERVER_NO_ERR
QMI_CORE_SERVER_INVALID_OBJECT
QMI_CORE_SERVER_MEMORY_ERR
QMI_CORE_SERVER_INTERNAL_ERR