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.

QUALCOMM® MSM™ Interface (QMI) Architecture

80-VB816-1 A

August 8, 2006

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

QUALCOMM® Proprietary

Restricted Distribution: This document contains critical information about QUALCOMM products and may not be distributed to anyone that is not an employee of QUALCOMM, its affiliates or subsidiaries without the approval of Configuration Management.

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

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document describes the QUALCOMM® MSM™ Interface (QMI) architecture and framework. The QMI allows applications on attached Terminal Equipment (TE) devices to access various Services provided by devices based on QUALCOMM's MSM chipsets and AMSS software.

1.2 Scope

It is expected that applications on a tethered computing device, e.g., connection manager applications and device drivers, will interface to the QMI-enabled MSM device using the QMI Multiplexing Protocol (QMUX) and QMI Service protocols as described herein.

1.3 Conventions

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

1.4 Revision history

The revision history for this document is shown in Table 1-1.

Table 1-1 Revision history

Version	Date	Description
A	Aug 2006	Initial release

1.5 References

Reference documents, which may include QUALCOMM, standards, and resource documents, are listed in Table 1-2. Reference documents that are no longer applicable are deleted from this table; therefore, reference numbers may not be sequential.

Table 1-2 Reference documents and standards

Ref.	Document	
QUALCOMM		
Q1	*QUALCOMM® MSM™ Interface (QMI) Global Constant Definitions*	80-VB816-2
Q2	*QUALCOMM® USB Network Driver for Windows® XP® User Guide*	80-VB717-1
Q3	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q4	*QMI Control Service (QMI_CTL)*	80-VB816-3

1.6 Technical assistance

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

If you do not have access to Internet web browsing, you may send email to support.cdmatech@qualcomm.com.

1.7 Acronyms

For definitions of terms and abbreviations, refer to [Q3].

2 QMI Framework

The QMI framework defines an interface between the TE and a processor running AMSS, enabling applications on the tethered processor to make use of functionality on the AMSS processor.

The QMI framework is composed of:

- Properties of the interconnection between an MSM chipset and the TE, including orthogonal control and data channels
- An enumeration of logical devices emulated by the MSM device over the interconnection
- A messaging protocol for messaging on the control channels of each logic device that allows applications running on the TE to access MSM-based Services

Figure 2-1 illustrates the layering of the QMI between the applications executing on a TE device and the MSM device.

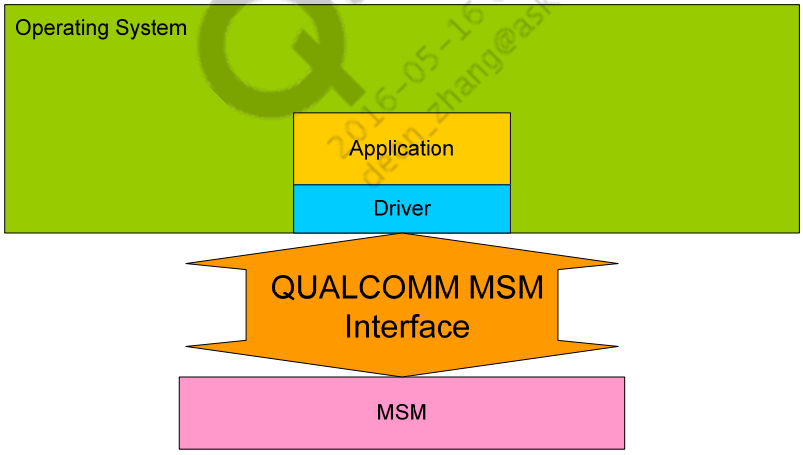

Figure 2-1 QUALCOMM MSM interface

The application-driver interface is described further in [Q2].

2.1 MSM-TE interconnection

QMI connects an MSM device to the TE (see Figure 2-2). The term TE is inclusive of all form factors, including devices such as PCs, notebooks, PDAs, and smartphones. The TE consists of an application environment (and possibly an operating system) executing on a separate processor, which is connected to the MSM processor via some form of interconnect.

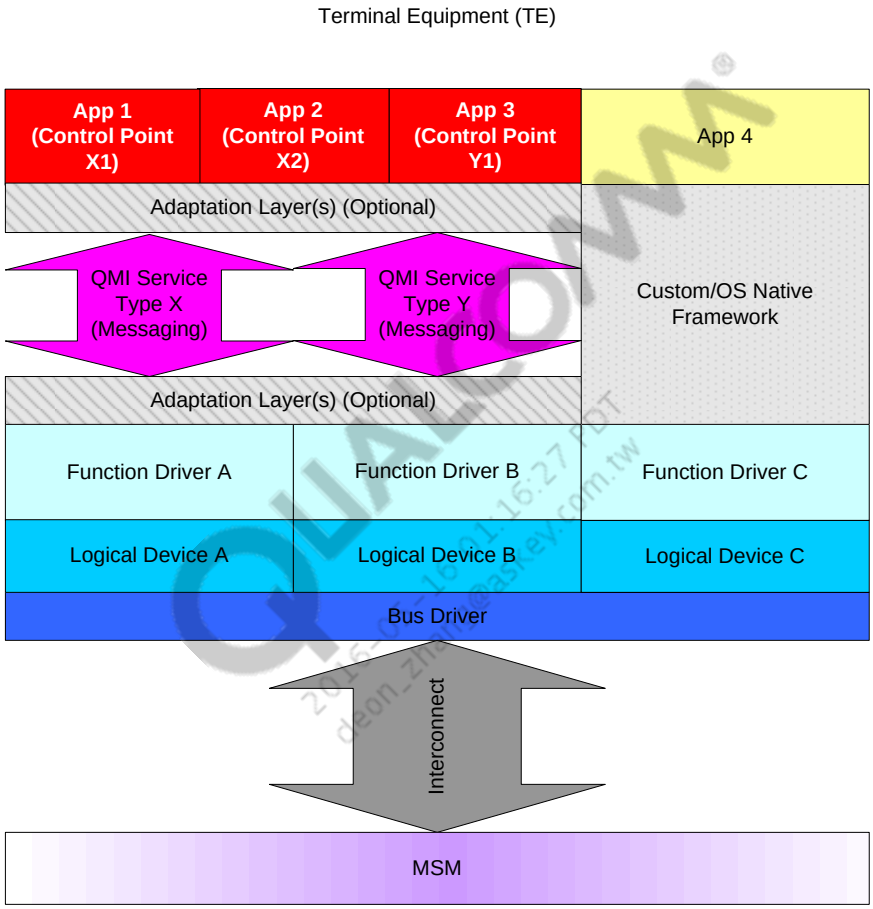

Figure 2-2 QMI architecture

The TE can be attached to the MSM over various bus interconnects, e.g., serial buses like USB, RS-232, PCI, or PCMCIA; wireless links like Bluetooth® or 802.11; shared memory interfaces, etc.

Regardless of which interconnect is used, QMI enumerates a number of logical devices. The interconnection must provide a mechanism for multiplexing multiple logical devices over a single physical connection.

Each logical device consists of at least one communication channel, and the underlying interconnect must provide for independent data and control communication channels for each logical device. Channel independence implies that each channel must act as if there were no physical coupling between the communication channels, including (but not limited to) separate Tx and Rx path queuing, independent flow control mechanisms, and independent data transmission scheduling.

A logical device uses at least one communication channel but need not have both (see Figure 2-3). For example, the existing MSM diagnostic interface consists of a data channel only.

For both QMI control and data channels, the interconnection must provide for framing of messages exchanged, i.e., delineating packet boundaries to the transport protocol (e.g., 802.3).

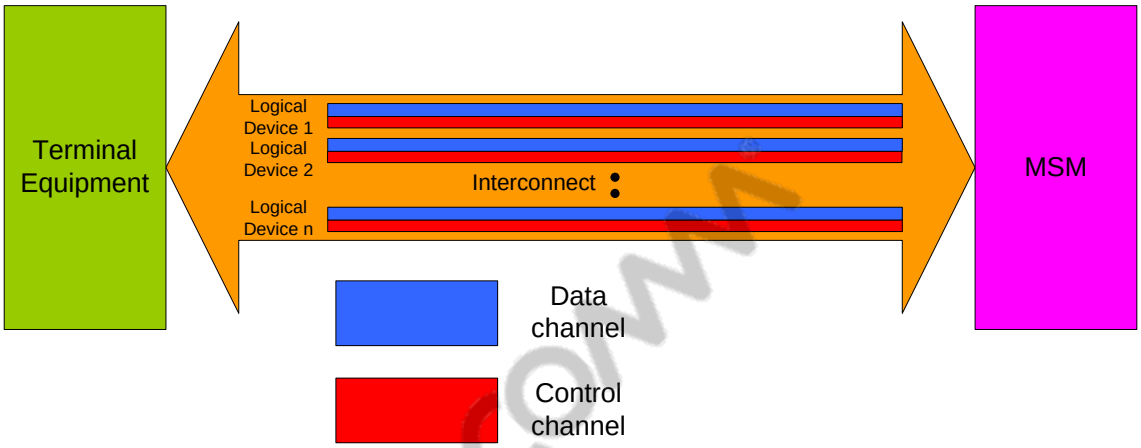

Figure 2-3 QMI MSM-TE interconnection

2.2 Logical device enumeration

Logical devices include both those that leverage QMI messaging protocols, such as an Rm network (RmNet) device.

Existing non-QMI devices are enumerated as well, such as:

- Legacy modem device
- Diagnostic interface
- NMEA device

Each logical device that is capable of exchanging QMI messages must provide orthogonal data and control channels. QMI messages are exchanged on the control channel.

The RmNet device presents an IP network interface to the TE provided by the wireless data-enabled QMI device.

The QMI logical device enumeration is detailed in [Q2].

2.3 Control channel messaging protocol

The QMI defines the protocol for communication over the control channel of a QMI logical device, consisting of:

- The QMUX transport protocol, which carries all control channel messages, described in Chapter 3 of this document
- A communication reference model defining communication endpoints known as Control Points and Services, described below

- A generalized QMI Service protocol, including protocols to be observed and rules for message definition, as described in Chapter 4; all QMI Service interfaces, including Services that conform to this generalized Service protocol and also custom QMI Services, are outside the scope of this document and are described in detail in their own specification document
- A special QMI_CTL Service that is used by the QMI drivers on both the TE and MSM devices to negotiate client IDs and special control Services; QMI_CTL conforms to general QMI Service protocol and is described in [Q4]

2.3.1 Endpoint model

Applications and device drivers on the TE communicate with a QMI-enabled MSM device by exchanging QMI Service messages over the QMUX transport protocol (see Chapter 3). These control messages are sent on the control channel of a QMI logical device (see Figure 2-4).

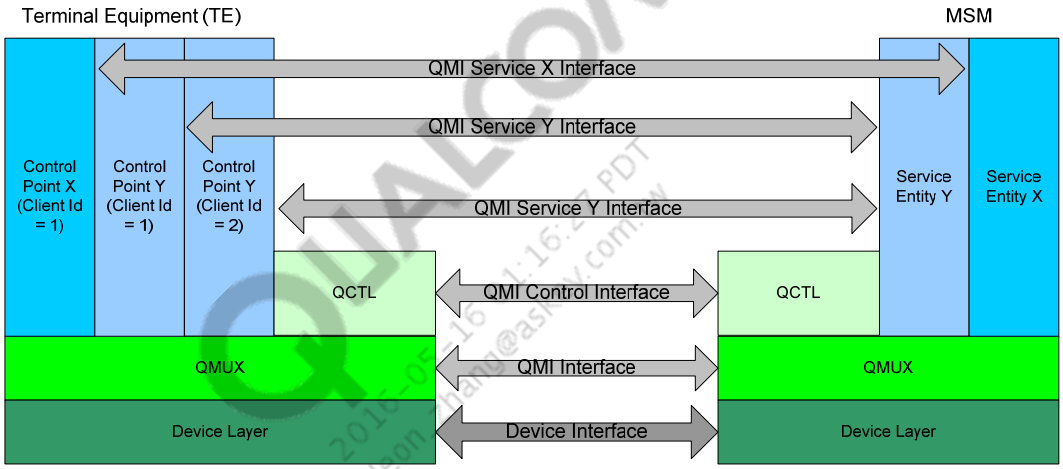

Figure 2-4 QMI control channel messaging endpoint model

All controlling applications are referred to as *Control Points*. A Control Point is a client of a particular QMI Service.

The software module that receives the QMI Service message and performs the function is referred to as a QMI Service.

A Control Point is to the Service as a client is to a server in the standard software engineering client/server model.

If an application makes use of several QMI Services, it will comprise a Control Point for each of the utilized Services.

2.4 Usage

Connection manager applications and device drivers on the TE are expected to interface to the QMI-enabled MSM device using QMI Service protocols.

Other applications on the TE may also be capable of using QMI Services.

3 QMI Multiplexing Protocol (QMUX)

A QMUX implementation provides transport for QMI Service messages between QMI Control Point(s) and QMI Service(s) over the control channel of a QMI logical device.

3.1 Byte ordering

QMUX messages shall be transmitted in little-endian format. The convention for bit numbering is that bit 0 is the Least Significant Bit (LSB).

3.2 QMUX message format

All QMUX messages employ a common header format, regardless of the direction in which the message is sent (Control Point to Service, or vice versa).

The QMUX message format is shown in Figure 3-1. Note that all messages sent on the control channel have a preamble byte (I/F type), which identifies the transport format (i.e., QMUX) (see Table 3-1).

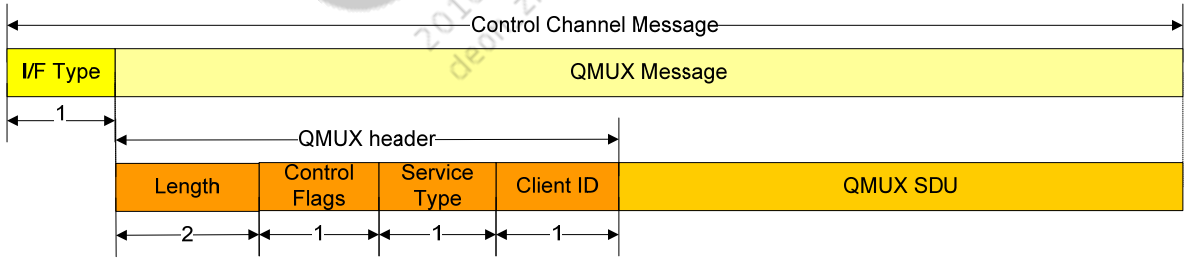

Figure 3-1 Control channel message format

Table 3-1 Control channel message preamble

Control channel message preamble	Description
I/F type	■ Interface type ■ 0x01 indicates that a QMUX message follows. ■ Other I/F type values are reserved for future transport protocol specifications.

The QMUX message consists of a header followed by an SDU containing the transported Service layer message(s).

The fields of the QMUX header are described in Table 3-2.

Table 3-2 QMI message fields

QMUX header field	Description
Length	Length of the QMUX message; includes the QMUX header itself, but not the preamble I/F type
Control flags	bit 7 – Sender type ■ 1 = Service ■ 0 = Control Point The remaining bits are reserved for future use and must be set to 0.
Service type	The QMI Service type of the message(s) present in the SDU. QMI Service types are defined in [Q1]. Individual licensees may define vendor-specific Services particular to their device(s) and assign them a Service type within the vendor-specific range [Q1].
Client ID	Identifies the Control Point to which the message pertains. Both the Service and Control Point should set this to the relevant Control Point client ID. A client ID value of 0xFF indicates a broadcast message that pertains to all Control Points of the preceding QMI Service type.[1]
QMUX SDU	Payload of the QMUX message; contains the QMI Service message(s)

[1] A Control Point shall not send a message with a client ID value of 0xff.

3.2.1 QMUX version

The QMUX protocol version will be distinguished by I/F type preamble field value.

3.2.2 Multiplexing of QMI Services

The QMUX multiplexes messages pertaining to different QMI Services by calling out the Service to which the transported message pertains. The QMUX enables multiplexing of messages corresponding to different Control Points by identifying the corresponding client ID in each message sent over the control channel. The QMUX also identifies the sending endpoint type (Control Point or Service).

Each QMI Service is assigned a unique QMI Service type identifier, defined in [Q1].

3.3 Rules for QMUX message handling

3.3.1 Generating QMUX messages

A QMUX protocol implementation:

- Appends a preamble and QMUX header to outgoing QMI Service messages, including the originating or destination Control Point's allocated client ID and the QMI Service type to which the message corresponds
- Sends the QMUX message over the control channel of the associated logical device

3.3.2 Receiving QMUX messages

A QMUX protocol implementation:

- Removes the preamble and QMUX header from incoming QMI Service messages
- Delivers the messages to the appropriate Service or Control Point based on the QMI Service type and current client ID mappings
- Provides an application adaptation layer (e.g., application API) to allow the application to send and receive Service messages and obtain/release client IDs

If the QMI Service type does not correspond to a Service implementation on the receiving device, the QMUX layer will discard the message.

3.4 Client ID management

Applications obtain/release client IDs from the QMI device driver on the host via the application adaptation layer. The details of this API are particular to the QMUX implementation and are thus outside the scope of this document. The reference implementation provided by QUALCOMM is described in [Q2].

The QMUX layer implementations in both the device and host driver implement the QMI control messages [Q4], which provide client ID management functions. These are used to implement the application client ID API.

4 QMI Generalized Service Message Protocol

This chapter describes the generalized message format and procedures that QMI Services should follow to ease implementation.

If a particular QMI Service diverges from this protocol, the corresponding QMI Service specification will document the superseding message format and/or procedures for that particular Service.

4.1 Service PDU message format

This section describes the message format generally followed by QMI Services.

4.1.1 Byte ordering

Numeric data in QMI Service messages shall be transmitted in little-endian format.

4.1.2 QMI transaction structure

A QMI Service sends and receives QMI transactions. A QMI transaction contains one or more QMI Service messages.

Figure 4-1 describes the format of a single QMI Service transaction.

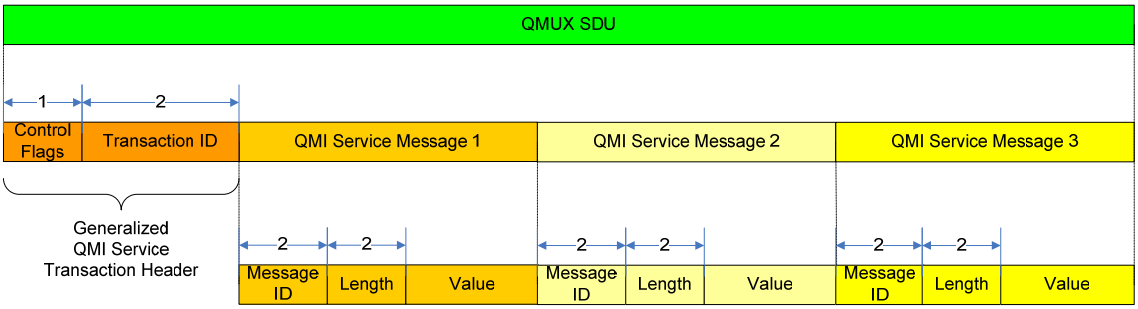

Figure 4-1 Generalized QMI Service transaction format

A QMI Service transaction starts with the general Service transaction header followed by one or more Service messages of the same QMI Service type.

The structure allows multiple Service messages of the same QMI Service type to be “bundled” together in the same transaction and dispatched in the same QMUX SDU.

The lower layer providing transport for the transaction (QMUX) delivers the transaction as a whole to the QMI peer (Control Point or Service).

The format of the generalized QMI Service message header is described in Table 4-1.

Table 4-1 QMI Service transaction header

<table>
<tr><th colspan="2">QMI Service header field</th><th>Offset (bytes)</th><th>Length</th><th>Description/values</th></tr>
<tr><td rowspan="3">Control Flags</td><td>Reserved</td><td>0</td><td>Bit 0</td><td>Must be 0</td></tr>
<tr><td>Message type</td><td>0</td><td>Bits 1-2</td><td>Message type for QMI Service message(s) present in the transaction:
<table>
<tr><th>Bit 2</th><th>Bit 1</th><th>Message type</th></tr>
<tr><td>0</td><td>0</td><td>Request</td></tr>
<tr><td>0</td><td>1</td><td>Response</td></tr>
<tr><td>1</td><td>0</td><td>Indication[1]</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
</table>
Message types are described in Section 4.2</td></tr>
<tr><td>Reserved</td><td>0</td><td>Bits 3-7</td><td>Must be 0</td></tr>
<tr><td colspan="2">Transaction ID</td><td>1</td><td>2 bytes</td><td>The transaction ID is an identifier used to distinguish transactions. Transaction IDs are unique among transactions for a particular Control Point.
The Control Point must increment the transaction ID each time a new transaction is sent. The transaction containing the response will use the same transaction ID. Each Control Point maintains its own transaction ID counter. This must always be a nonzero value.</td></tr>
</table>

[1] Indications cannot be bundled.

4.1.3 QMI message structure

A single QMI Service message is formatted as described in Figure 4-2.

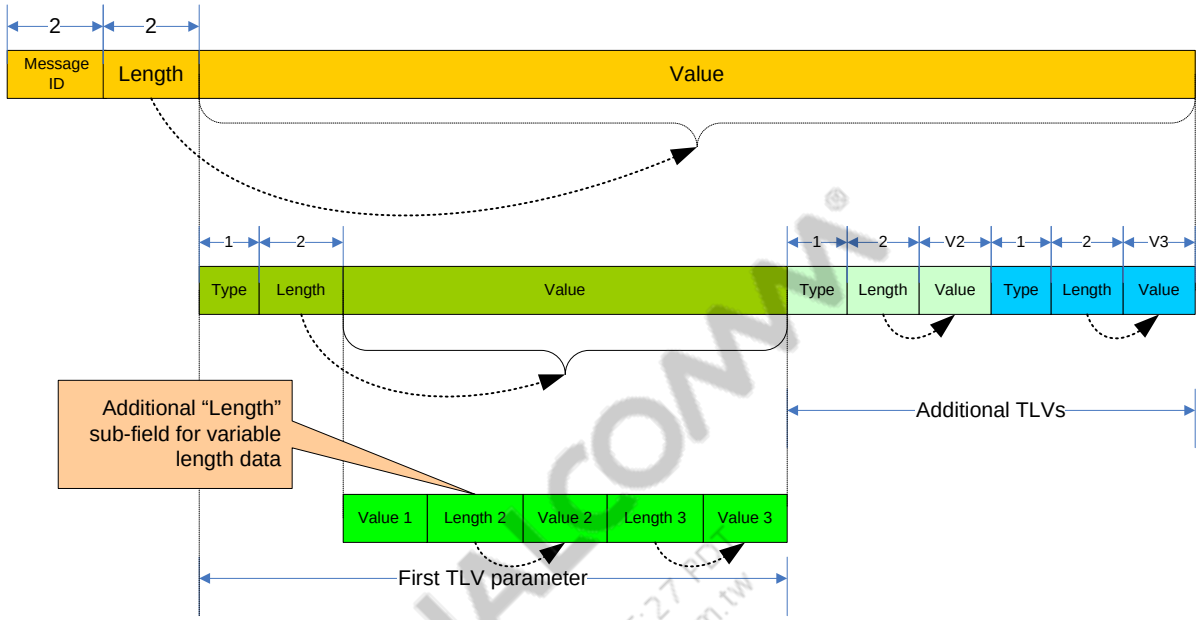

Figure 4-2 Generalized QMI Service message and parameter formats

The QMI Service messages are distinguished by QMI message ID.

Each QMI Service type has its own set of QMI messages, defined in the QMI Service specification document. See [Q1] for a current list of QMI Service specifications that have been defined.

The same message ID value is used in corresponding request and response messages. If an indication message is defined corresponding to the request and response, it will share the same message ID value.

The length field following the message ID indicates the total number of bytes in the message following the length field, i.e., the total length of all parameters included in the message.

The value portion of the message consists of zero or more parameters associated with the message. The value typically contains the information required to execute the requested action or results of the action.

4.1.4 QMI message parameter structure

Figure 4-2 illustrates parameters within the value portion of the QMI message.

Message parameters are defined separately for each request, response, or indication message. Message parameters are formatted with three sections, type, length, and value. Because of this, message parameters are sometimes referred to as TLVs.

4.1.4.1 Parameter types

The parameter type field indicates which parameter is being specified.

A unique TLV parameter type is defined for each parameter that may be specified within a given message type. The same parameter type may have a different meaning in the context of other messages.

4.1.4.2 Parameter length

The parameter length indicates the length in bytes of the following value field.

The expected length will be documented per parameter in the QMI Service specification. This will be a fixed value when the value field is a fixed structure. If the parameter contains a string or other variable-length data, this will be defined as a calculated value. For example, if the value section includes a variable length string, the length field will tell the receiver how many bytes are in that string.

4.1.4.3 Parameter value

The value of a parameter contains the actual information communicated by including the parameter in the message.

The entire parameter, as defined in the Service specification, must be present. Any flexibility in format of the value portion of the parameter will be described in the parameter value description.

All numeric data are positive (unsigned) binary values unless stated otherwise in the parameter description.

4.2 QMI message types

The generalized QMI Service transaction format defines three basic message types. All three message types follow the generalized QMI Service message format described in Section 4.1.3.

4.2.1 Request

A request message may be used to set parameters, query parameter values, or configure the generation of indications.

The request message is issued by the Control Point.

A valid request always generates a response from the Service.

4.2.2 Response

A response message is issued by the Service, in response to a received request.

Each response contains at least the result parameter indicating that the request succeeded or failed, and the error status, indicating the result of the operation requested. Additional parameters may be present to communicate data associated with the operation.

4.2.3 Indication

An indication is sent by a QMI Service to inform Control Point(s) of changes in state.

The indication message is issued by the Service without any solicitation by a Control Point.

4.2.3.1 Unicast vs. broadcast indications

Indications from the Service are either broadcast to all Control Points or unicast to a specific Control Point. Indication type is indicated by the value of the client ID field in the QMUX header.

The definition of the indication message (in the associated QMI Service specification document) specifies whether it shall be unicast or broadcast.

4.3 Rules for transaction handling

A generalized QMI Service receives a QMUX SDU from the QMUX layer when the QMUX header identifies the Service type corresponding to that Service. This QMUX SDU contains a single QMI Service transaction.

When a request transaction is received, the QMI Service must:

- Remember the transaction ID and use that same value in the transaction ID of the corresponding response message sent back to the Control Point
- Process all requests in a transaction, in the order specified; all requests in the bundle will be processed, regardless of the result of preceding bundled request(s)
- Return all corresponding responses in the same order as the requests in a single response transaction (i.e., sent in the same QMUX SDU); no additional messages may be added to this response transaction
- Send the response transaction only when all requests are complete

If any message length field (per the message header) in the transaction indicates data past the end of the transaction (per the transaction header), the entire transaction will be considered malformed and none of the contained requests will be processed. This is done since it is ambiguous to the receiver whether the message indicating the invalid data length is corrupt or if a previous message was corrupt. This may lead the receiver to misinterpret data as a subsequent message header.

If the Service cannot process the transaction because too many QMI messages were contained, none of the requests will be processed. This is done to avoid partially executing the Control Point's requests. Unless otherwise specified in the QMI Service specification, a generalized QMI Service will support up to five request messages in a transaction. This may be further limited by outstanding message limitations described in Section 4.4.

Note that when multiple requests are contained in a transaction, the response will be delayed until all commands have been processed. Hence, if the processing/execution of one request contained in such a transaction is much slower than the other requests, the Control Point will not get any results until the slowest message has executed.

If a Control Point wishes to execute a request conditional to the result of a preceding request, it should issue each in a separate transaction.

4.4 Rules for message handling

4.4.1 General rules

All messages generated by that QMI Service and its Control Points must set the Service type field in the QMUX header to the QMI Service type assigned to that Service.

The receiver shall verify that the message and any parameters are formatted correctly before processing the message.

Unless otherwise specified in the message definition, messages apply to all QMI devices, independent of the wireless technology employed by the device.

All QMI Service messaging is asynchronous in that the receiver must not expect a response to be returned immediately following the issuing of the request. Further, a Control Point may issue a subsequent request before the response to the first is received. Indications and responses generated by the Service may be sent in any order.

4.4.2 Generating requests

Control Points will formulate requests according to the general Service message protocol or any superseding protocols defined in the QMI Service specification document.

4.4.3 Receiving requests

If the length of a parameter specified in the received request is incorrect, the Service will be unable to process the request. In this case, the Service will fail the request with QMI_ERR_MALFORMED_MSG.

Unless otherwise specified in the QMI Service specification, a particular client is limited to five outstanding requests at a given time. An outstanding request is one for which a QMI response has not been received by the Control Point (not including discarded requests, e.g., requests in malformed transactions).

Typically, the QMI Service will generate and send the response immediately while processing the request message and executing the command. However, some actions (such as a network scan) take some time in which case the sending of the response will be delayed until the action is complete.

4.4.4 Generating responses

For each QMI request received, the response shall include a result parameter that indicates success or failure and an error code. If failure is returned, an error status value is returned in the result parameter and none of the associated mandatory parameters will be included in the response, since there is no valid information to return. The set of possible error values returnable for each command is defined in the Service specification document. The QMI error constant definitions are documented in [Q1].

Additional TLVs may be included on a per-message basis to communicate required information, e.g., queried device parameter.

If an unrecognized request is received by the Service, it may return a response using the unrecognized message type, and a single result TLV indicating result failure and error code QMI_ERR_UNRECOGNIZED_REQUEST.

4.4.5 Receiving responses

A QMI Control Point must be prepared to receive a response some time after the request is sent. The Control Point must be prepared to receive a responses to requests in a different order than the requests are issued. Further, the Control Point must be able to process indication messages at any time; i.e., it must not expect a response to be the next message received after the request message is sent.

4.4.6 Generating indications

A Service may broadcast a single indication message across all registered Control Points by setting the QMUX header client ID field to 0xFF. Broadcast indications are defined for Service status that is of interest to all Control Points, reducing the QMI link bandwidth needed to deliver that information to all.

When the QMUX layer on the receive side receives a broadcast indication, it must generate and deliver one copy of the received broadcast indication to each Control Point registered to that Service.

Unicast indications are addressed to a particular Control Point by setting the QMUX client ID field to the intended Control Point's client ID value. Unicast indications convey status particular to a single Control Point.

4.4.7 Receiving indications

The Control Point may take action based on the contents and/or update any associated user interfaces.

The Control Point shall ignore any received indications that are unrecognized or unsupported.

The transaction ID field of a transaction containing an indication message should be ignored by the Control Point.

4.5 Rules for parameter handling

Messages may have mandatory and/or optional parameters associated with them. Unless explicitly specified in the definition of the message, there is no particular ordering requirement for parameters.

NOTE The parameters are specified in the general format of [type|length|value] and hence are sometimes referred to as TLVs.

Mandatory parameters (TLVs):

- Must be included in the message by the sender in all cases
- Are assigned type field values less than 0x10

Optional parameters (TLVs):

- Are generated and/or processed at the Service or Control Point at their discretion; i.e., need not be implemented unless otherwise specified
- Are assigned type field values greater than 0x0F
- Must be ignored by the receiver if unrecognized/not implemented
- Must not be relied on by the receiver; i.e., if optional parameters are not specified in a response or indication message, this should not cause an error or disrupt proper operation of the Control Point
- May be conditionally required to be included on a per-message basis, as described in the message definition in the appropriate QMI Service specification

4.6 State variables

QMI Services may keep track of state related to the internal functionality accessed through that Service in Service global state variables. The Service may also keep track of Control Point settings and state in Control Point state variables.

When a Control Point is allocated a new client ID, and when that client ID is released, that client ID's state variables are set to the default settings.

Upon powerup, and when the QMI link is disconnected, Service global state variables are reset to their default settings.

The handling of state variables and their impact on the system is described in the QMI Service specification document.

4.7 Control Point arbitration

It is possible to have multiple Control Points interact with a single Service on the QMI device.

In cases where multiple Control Points issue messages related to a common resource, the default policy is that the actions will be executed in the order received; hence, the "last request wins."

In some cases, more careful arbitration of a common resource is managed by keeping track of Control Point requests via state variables. In such cases, the message definition may describe any the arbitration policy for the common resource.

4.8 QMI Service versioning

QMI Control Points and QMI Services are written to a particular version of a QMI Service specification document. Since the Service specifications are compiled to over time, Control Points may want to know the Service version implemented on a device, to know whether specific functionality within the Service is supported.

4.8.1 Version format

Each QMI Service has its own version number that is independent of other QMI Services. A QMI Service version is represented as M.n where:

- M = major version, 2 bytes
- n = minor version, 2 bytes

4.8.2 Learning QMI Service versions

The QMI driver on the TE provides an API to learn the Service version [Q2].

4.8.3 Service versioning rules

4.8.3.1 Major versions

As the major version of the Service is incremented, the Service specification is changed in a way that breaks backward compatibility with the previous version.

Control Points should not assume interoperability with a Service that has a different major version.

A QMI Service is required only to support one major version of a QMI Service. A QMI Service may implement multiple major versions of a QMI Service.

4.8.3.2 Minor versions

The minor version of the Service is incremented when the Service specification is modified without breaking backward compatibility with previous versions sharing the same major revision number.

Control Points may assume interoperability with a Service that has a different minor version.

4.8.4 Message and parameter updates

Each message definition will indicate the QMI Service version in which it was first defined. The Control Point should consider this the minimum required Service version to carry out the operation associated with that request.

Each parameter definition will indicate the QMI Service version in which it was last modified. Since the backward compatibility requirement implicit to QMI ensures that parameters will not be changed in a way that renders an older minor revision incompatible, it is not critical for the application to take action based on the last modified version. This is provided as a quick means for the application writer to identify updated fields in a newer Service specification that might be handled by the application; however, the application will work without implementing any of these changes.