



# Qualcomm® Unified Tool Service (QUTS)

## User Guide

### INTERNAL USE ONLY

80-PG522-3 Rev. A

July 30, 2019

**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](mailto:DocCtrlAgent@qualcomm.com).

**Restricted Distribution:** Not to be distributed to anyone who is not an employee of either Qualcomm Technologies, Inc. or its affiliated companies without the express approval of Qualcomm 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.

All Qualcomm products mentioned herein are products of Qualcomm Technologies, Inc. and/or its subsidiaries.

Qualcomm is a trademark of Qualcomm Incorporated, registered in the United States and other countries. QXDM Professional is a trademark of Qualcomm Incorporated. 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.

© 2020 Qualcomm Technologies, Inc. and/or its subsidiaries. All rights reserved.

## Revision history

Revision	Date	Description
A	July 2019	Initial release

# Contents

---

<b>1 Introduction.....</b>	<b>5</b>
1.1 Purpose .....	5
1.2 Conventions .....	5
1.3 Technical assistance .....	5
<b>2 Functional overview .....</b>	<b>6</b>
<b>3 Use cases .....</b>	<b>7</b>
3.1 Using QUTS .....	7
3.2 Set up a connection to QUTS .....	7
3.3 Use the service .....	7
3.4 Postprocessing log sessions .....	8
3.5 Field queries .....	8
<b>4 QUTS application interfaces .....</b>	<b>9</b>
4.1 QutsClientManager interface .....	9
4.1.1 Constants .....	9
4.1.2 Types .....	9
4.1.3 Functions .....	10
4.2 DeviceManager interface .....	12
4.2.1 Types .....	12
4.2.2 Functions .....	46
4.3 ClientCallback interface .....	67
4.3.1 Types .....	67
4.3.2 Functions .....	69
4.4 LogSession interface .....	76
4.4.1 Types .....	76
4.4.2 Functions .....	81
4.5 UtilityService interface .....	89
4.5.1 Types .....	89
4.5.2 Functions .....	90
<b>5 QUTS services interfaces .....</b>	<b>93</b>
5.1 Device config service .....	93
5.1.1 Constants .....	93
5.1.2 Types .....	93
5.1.3 Functions .....	98
5.2 Diag service .....	118
5.2.1 Types .....	119

5.2.2 Functions .....	120
5.3 Image management service .....	131
5.3.1 Types .....	132
5.3.2 Functions .....	134
5.4 QMI service .....	139
5.4.1 Functions .....	139
5.5 Raw service .....	146
5.5.1 Constants .....	146
5.5.2 Types .....	146
5.5.3 Functions .....	147
5.6 ADB service .....	151
5.6.1 Functions .....	151
<b>A References .....</b>	<b>158</b>
A.1 Acronyms and terms .....	159

# 1 Introduction

---

## 1.1 Purpose

This document describes the Qualcomm® Unified Tools Service (QUTS) interfaces. Included are an overview of QUTS, call flows, and descriptions of the APIs.

## 1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `cp armcc armcpp`.

Code variables appear in angle brackets, for example, `<number>`.

Commands to be entered appear in a different font, for example, `copy a:*. * b:.`

## 1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at <https://createpoint.qti.qualcomm.com/>.

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

## 2 Functional overview

---

QUTS provides access to all Qualcomm devices connected to the computer on which it's running. The single instance of the QUTS service provides device discovery and access control to the different interfaces made available through Qualcomm devices. Client applications will then connect to QUTS to access functionality provided over these connections.

Application clients must communicate with QUTS using the socket-based RPC system, Apache Thrift. For some languages (such as C, C++, PERL, etc.), a wrapper for this communication layer is provided along with the application. For other languages supported by Thrift, the communication layer must be developed by the client.

QUTS provides some callback mechanisms to which clients can optionally connect. These interfaces provide valuable runtime information or provide a mechanism for querying input for QUTS. This interface is described in Chapter 4. To connect to these functions, a Thrift server must be made available from the client application. If you are using a provided wrapper, this is done automatically.

## 3 Use cases

---

### 3.1 Using QUTS

For supported languages (C++, C#, Python, Perl), there is a QutsClient module distributed with QUTS that allows easy connection and configuration with QUTS. Create an instance of the QutsClient then call `getDeviceManager()` to return an instance of the DeviceManager (see Section 4.3). If using any other language supported by Thrift, the language bindings must be generated using `thrift.exe`, then following the procedure in Section 3.2.

### 3.2 Set up a connection to QUTS

1. Call `registerClient()` on UTS\_REGISTRATION\_PORT (50090) using the *QUTS Registration* Thrift service.

This operation allocates a client service port, which is returned from the function call. All subsequent calls are done on that service port via the DeviceManager server using the *Device Manager* Thrift service.

2. (Optional) Start up and register a client callback server to handle notifications and input requests from QUTS.

Run this operation on the port (client port + 1). It must implement a Thrift multiplexed server with a single service called *QUTS Callback*.

3. Make function calls to the DeviceManager.

If an error value is returned, call `getLastError()` to determine the reason for the failure. For usage errors, QUTS throws an `AppException` that contains information regarding the error. For device errors, QUTS provides failure information in either the function return type or through the `getLastError` interface.

### 3.3 Use the service

At this point the service is ready for use. A typical situation makes use of services or a connection to a device. Following is a typical call flow for using a device:

1. Call `getServicesList()` to get a list of available services, and choose the service to be used.
2. For a list of active devices that support a specified service, call `getDevicesForService()` and indicate the `serviceName`.
3. To start an instance of the service communicating with a specified device, call `createService()` and indicate the `serviceName` and `deviceHandle`.
4. Create a Thrift service client using the name returned from `createService()`.

Each service has its own API (see Chapter 4.5.2.1).

For services provided by QUTS, modules are provided to simplify the process of service creation. C++ DLLs, PERL modules, and Python modules are distributed with QUTS to facilitate this process. For usage ideas, see the samples provided in this document.

### 3.4 Postprocessing log sessions

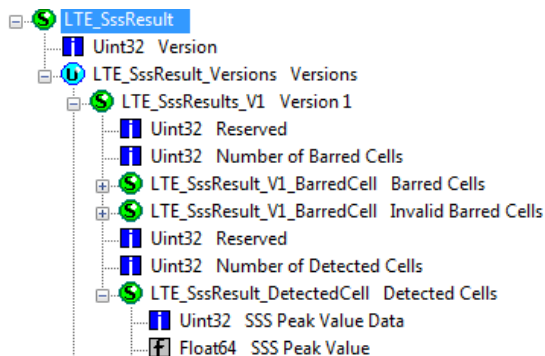
Alternative to using services on live connections, QUTS can load and iterate log files for postprocessing as follows:

1. Call `openLogSession()` (see Section 4.3.2.21) with a list of log files that were generated in a single session of live traffic in QUTS.
2. Create a Thrift `LogSession` service client using the name returned from `openLogSession` (see Section 4.3.2.21).

Multiple log sessions can be created and processed independently. Each log session has an API as defined in Section 4.4.2.15.

### 3.5 Field queries

Field queries allow for the access of specific field values from structures. The user can provide partial paths or fully qualified paths based on the structure of the data packet containing the fields. For example, given the following structure:



The following table provides example accessors.

Accessor	Description
"SSS Peak Value"	Matches every instance of "SSS Peak Value" in the packet
"SSS Peak Value"[0]	Matches only instances of "SSS Peak Value" at array index 0
"Detected Cells"."SSS Peak Value"	Matches only instances "SSS Peak Value" that exist within a field called "Detected Cells"
"Detected Cells"[1]."SSS Peak Value"	Matches only instances of "SSS Peak Value" that exist within a field called "Detected Cells" where "Detected Cells" is at index 1



# 4 QUTS application interfaces

## 4.1 QutsClientManager interface

### 4.1.1 Constants

#### 4.1.1.1 UTS\_REGISTRATION\_PORT

Static port that QUTS uses for registering new clients. Only `registerClient()` is called on this port.

```
const i32 UTS_REGISTRATION_PORT = 50090
```

### 4.1.2 Types

#### 4.1.2.1 ClientPortOffsets

Provides the offsets for ports relative to the port number allocated by the QUTS registration by `registerClient()`.

```
enum ClientPortOffsets
{
    PORT_DEVICE_MANAGER_SERVER = 0,
    PORT_CALLBACK_CLIENT = 1
}
```

PORT_DEVICE_MANAGER_SERVER	Port offset from <code>registerClient()</code> on which DeviceManager calls will be running.
PORT_CALLBACK_CLIENT	Port on which the ClientCallback server should be instantiated by the client application.

### 4.1.3 Functions

#### 4.1.3.1 registerClient()

Allocates a new DeviceManager server for servicing client function calls.

##### Prototype

```
i32 registerClient()
```

##### Returns

Port number on which the new DeviceManager is set up.

## 4.2 Scripting Interface methods

For few specific languages (Python, Perl, C# and Java), QUTS provides wrapper APIs that make starting QUTS client easier by hiding all thrift calls. As an example, the initial calls on Python are described here. To get a QutsClient object, you can do:

```
import QutsClient
import Common.ttypes

client = QutsClient.QutsClient("QUTS Sample", [multithreadedClient])
```

If multiple threads are needed in the script side, set `multithreadedClient` to `True`. Default value is `False`.

Note: If `multithreadedClient` is set to `True`, then call `stop()` on `qutsClient` object at end of testcase is needed for proper cleanup. If `qutsClient` is created in main, then call to `stop()` is needed (if `multithreadedClient` set to `True`). However, if `qutsClient` is created in a method which is called from main, then when that method exits, `stop()` is automatically called when `qutsClient` goes out of scope/destroyed, so explicit call to `stop()` is not needed in this case.

### 4.2.1 QutsClient Interface Functions

#### 4.2.1.1 getDeviceManager

Gets the deviceManager handle. All other methods defined in the deviceManager interface can be called on this handle.

#### 4.2.1.2 getUtilityService

Gets the utilityService handle. All other methods defined in the utilityService interface can be called on this handle.

#### 4.2.1.3 openLogSession

Opens logSession. Same as openLogSession on DeviceManager handle.

#### 4.2.1.4 getActiveLogSession

Opens existing logSession. Same as getActiveLogSession on DeviceManager handle.

#### 4.2.1.5 createService

Used to create any service such as Diag/Qmi etc. As an example, diagService can be created as:

```
import DiagService.DiagService
import DiagService.constants
import DiagService.ttypes

diagService = DiagService.DiagService.Client(
    qutsClient.createService(DiagService.constants.DIAG_SERVICE_NAME,
                           deviceId))
```

### 4.2.2 Qutsclient interface call back methods

These are callback methods. When the corresponding event occurs, the callback is invoked. An example is provided for first one, rest of the callback methods usage is similar.

#### 4.2.2.1 setOnMessageCallback

e.g. `qutsClient.setOnMessageCallback(onMessageCallback)`

#### 4.2.2.2 setOnDeviceConnectedCallback

#### 4.2.2.3 setOnDeviceDisconnectedCallback

#### 4.2.2.4 setOnDeviceModeChangeCallback

#### 4.2.2.5 setOnProtocolAddedCallback

#### 4.2.2.6 setOnProtocolRemovedCallback

#### 4.2.2.7 setOnProtocolStateChangeCallback

#### 4.2.2.8 setOnMissingQShrinkHashFileCallback

#### 4.2.2.9 setOnLogSessionMissingQShrinkHashFileCallback

**4.2.2.10** setOnAsyncResponseCallback

**4.2.2.11** setOnDataQueueUpdatedCallback

**4.2.2.12** setOnDataViewUpdatedCallback

**4.2.2.13** setOnServiceAvailableCallback

**4.2.2.14** setOnServiceEndedCallback

**4.2.2.15** setOnServiceEventCallback

**4.2.2.16** setOnQShrinkStateUpdated

## 4.3 DeviceManager interface

### 4.3.1 Types

#### 4.3.1.1 ErrorCode

Reasons for a failed function call.

```
enum ErrorCode
{
    DEVICE_NO_ERROR = 0,
    DEVICE_UNKNOWN_ERROR,
    DEVICE_INVALID_PARAMETERS,
    DEVICE_PERMISSIONS_ERROR,
    DEVICE_INVALID_DEVICE_HANDLE,
    DEVICE_INVALID_PROTOCOL_HANDLE,
    DEVICE_INVALID_CONNECTION_HANDLE,
    DEVICE_CONNECTION_LOCKED,
    DEVICE_DISCONNECTED,
    DEVICE_PROTOCOL_INVALID,
    DEVICE_PROTOCOL_DISCONNECTED,
    DEVICE_PROTOCOL_UNRESPONSIVE,
    DEVICE_TX_CANCELLED,
    DEVICE_TX_TIMEOUT,
    DEVICE_INVALID_PROCESSOR,
    DEVICE_INVALID_PACKET,
    DEVICE_RESPONSE_ERROR,
    DEVICE_INVALID_LOG_SESSION,
    DEVICE_SERVICE_NOT_INITIALIZED,
    DEVICE_TCP_PORT_FAILURE,
    DEVICE_SERVICE_ALREADY_INITIALIZED
}
```

#### 4.3.1.2 ErrorType

Bundles the latest error code with a string that describes the error in more detail.

```
struct ErrorType
{
    1: required ErrorCode errorCode;
    2: required string errorString;
}
```

#### 4.3.1.3 AppException

This exception is thrown by QUTS when there is a usage issue or unknown error. The specific error information is set in the errorCode and errorString.

```
exception AppException
{
    1: required ErrorCode errorCode;
    2: required string errorString;
}
```

#### 4.3.1.4 ProtocolType

Type of interface that a protocol represents.

```
enum ProtocolType
{
    PROT_UNKNOWN = -1,
    PROT_DIAG = 0,
    PROT_QMI,
    PROT_ADB,
    PROT_SAHARA,
    PROT_FIREHOSE,
    PROT_QDSS,
    PROT_ADPL,
    PROT_FASTBOOT,
    PROT_NMEA,
    PROT_DUN,
    PROT_RMNET,
    PROT_RNDIS,
    PROT_MBIM,
    PROT_MAX
}
```

#### 4.3.1.5 ConnectionType

Type of physical connection that a protocol uses.

```
enum ConnectionType
{
    CONNECT_UNKNOWN = -1,
    CONNECT_USB = 0,
    CONNECT_TCP,
    CONNECT_FILE,
    CONNECT_ETHERNET,
    CONNECT_MAX
}
```

#### 4.3.1.6 OpenProp

Permissions for opening and sharing a connection.

```
enum OpenProp
{
    OPEN_NONE = 0x0000,
    OPEN_READ = 0x0001,
    OPEN_WRITE = 0x0002,
    OPEN_READ_WRITE = 0x0003
}
```

#### 4.3.1.7 DeviceMode

Device operation mode bitmask.

```
enum DeviceMode
{
    DEVICE_MODE_NONE = 0x00000000,
    DEVICE_MODE_SAHARA_DOWNLOAD = 0x00000001,
    DEVICE_MODE_SAHARA_CRASH = 0x00000002,
    DEVICE_MODE_SAHARA_EFS_SYNC = 0x00000004
}
```

#### 4.3.1.8 ProtocolInfo

Structure that contains all relevant information for a specific protocol.

```
struct ProtocolInfo {
    1: i64 protocolHandle;
    2: i64 deviceHandle;
    3: string description;
    4: ProtocolType protocolType;
    5: ConnectionType connectionType;
    6: OpenProp connectionStatus;
```

```

7: OpenProp shareStatus;
8: ProtocolState protocolState;
}

```

## Fields

protocolHandle	64-bit handle for referencing the protocol.
deviceHandle	64-bit handle for the device to which the protocol belongs.
Description	Human-readable text description of the protocol.
protocolType	Type of protocol (see <a href="#">AppException</a> ).  This exception is thrown by QUTS when there is a usage issue or unknown error. The specific error information is set in the errorCode and errorString.  <pre> exception AppException {     1: required ErrorCode errorCode;     2: required string errorString; } </pre> ProtocolType).
connectionType	Type of physical connection (see <a href="#">ConnectionType</a> ).
connectionStatus	Connection state of the protocol, if a service is using the protocol.
shareStatus	Indicates whether the protocol is available based on current sharing restrictions.
protocolState	Indicates the current protocol state (see <a href="#">ProtocolState</a> ).

### 4.3.1.9 DeviceInfo

Structure that contains all relevant information for a specific device.

```

struct DeviceInfo {
    1: i64 deviceHandle;
    2: string description;
    3: list<ProtocolInfo> protocols;
    4: list<string> services;
    5: string serialNumber;
    6: string adbSerialNumber;
    7: string location;
}

```

## Fields

deviceHandle	64-bit handle for referencing the device.
description	Human-readable text description of the device.
protocols	List of protocols on the device (see <a href="#">ProtocolInfo</a> ).
services	List of services currently available on the device.

serialNumber	MSM serial number if it is available.
adbSerialNumber	ADB serial number if it is available.
location	Hub location number of the device.

#### 4.3.1.10 DiagPacketType

Enumerates the different categories of diag payloads. Each has its own ID system below the diag level command code.

```
enum DiagPacketType
{
    LOG_PACKET = 0,
    EVENT,
    NV_ITEM,
    DEBUG_MSG,
    REQUEST,
    RESPONSE,
    SUBSYS_REQUEST,
    SUBSYS_RESPONSE,
    SUBSYSV2_REQUEST,
    SUBSYSV2_IMMEDIATE_RESPONSE,
    SUBSYSV2_DELAYED_RESPONSE,
    QTRACE,
    QSH_METRIC
}
```

#### Values

LOG_PACKET	Corresponds to command ID 16 (0x10). ID values are log codes, names are the names of the log packets (for example, 0x1234 or "Log Packet Name").
EVENT	Corresponds to command ID 96 (0x60). ID values are event IDs, names are the names of the events (for example, 123 or "Event Name").
NV_ITEM	Corresponds to NV read/set related commands or subsystem commands. ID values are NV IDs, names are either the name of the legacy NV item or the EFS path for the NV item (for example, 12 or "My NV Value" or "/nv/item").
DEBUG_MSG	Corresponds to command ID 121 (0x79). ID values are the combination of subsystem ID and Level (for example, 123/3 or "Subsystem/High").
REQUEST	Any outgoing command code other than those enumerated in this table. ID values are the command code, names are the request name (for example, 12 or "Command Request").
RESPONSE	Any incoming command code other than those enumerated in this table. ID values are the command code, names are the request name (for example, 12 or "Command Response").
SUBSYS_REQUEST	Corresponds to outgoing command ID 75 (0x4B). ID values are the combination of subsystem ID and subsystem command code (for example, 12/1 or "Subsystem/Command Name").



SUBSYS_RESPONSE	Corresponds to incoming command ID 75 (0x4B). ID values are the combination of subsystem ID and subsystem command code (for example, 12/1 or "Subsystem/Command Name").
SUBSYSV2_REQUEST	Corresponds to outgoing command ID 128 (0x80). ID values are the combination of subsystem ID and subsystem command code (for example, 12/1 or "Subsystem/Command Name").
SUBSYS_V2_IMMEDIATE_RESPONSE	Corresponds to the immediate incoming command ID 128 (0x80) responding to a SUBSYSV2_REQUEST. ID values are the combination of subsystem ID and subsystem command code (for example, 12/1 or "Subsystem/Command Name").
SUBSYS_V2_DELAYED_RESPONSE	Corresponds to subsequent incoming command ID 128 (0x80) after receiving the initial SUBSYSV2_IMMEDIATE_RESPONSE. ID values are the combination of subsystem ID and subsystem command code (for example, 12/1 or "Subsystem/Command Name").
QTRACE	Corresponds to command ID 157 (0x9D). ID values are the combination of client ID and buffer ID (for example, 12/1 or "Client Name/Buffer").
QSH_METRIC	Corresponds to log packet 0x19B6. ID values are the combination of client ID and metric ID (for example, 12/1 or "Client Name/Metric").

#### 4.3.1.11 DiagIdFilterItem

The `DiagIdFilterItem` allows for specification of an ID level filter. It is used in conjunction with [DiagPacketFilter](#). It specifies an optional ID and two optional regex-based filters for parsed text and summary text. If no ID is given, the regex is applied to all items of this type (for instance, all `DEBUG_MSG`). If no regex is given, filtering is done at the ID level.

```
struct DiagIdFilterItem
{
    1: optional string idOrName;
    2: optional string regexFilter;
    3: optional string summaryRegexFilter;
}
```

#### Fields

idOrName	Name or ID of the item. For ID definitions, see <a href="#">DiagPacketType</a> .
regexFilter	Regular expression to filter based on the parsed text of the item
summaryRegexFilter	Regular expression to filter based on the summary text of the item

#### 4.3.1.12 DiagPacketFilter

DiagPacketFilter provides the ability to specify which items are required from a Diag stream.

```
struct DiagPacketFilter
{
    1: optional map<DiagPacketType, list<DiagIdFilterItem> >
    idOrNameMask;
    2: optional list<i32> subscriptionId;
    3: optional InputLogMask logMask;
    4: optional list<string> formatStringFilter;
    5: optional bool enableMultiSim;
}
```

#### Fields

idOrNameMask	For each <a href="#">DiagPacketType</a> this can provide many ID/Regex combinations based on <a href="#">DiagIdFilterItem</a> . If a <a href="#">DiagPacketType</a> is not included or it has an empty list, it is not included in the results (it is filtered out).
subscriptionId	Allows specification of a subset of subscription IDs. If no ID is specified, all are passed (no filtering is done on the subscription ID).
logMask	Input log mask
formatStringFilter	Format string Filter
enableMultiSim	If the multisim mask is to be enabled, this needs to be set to true

#### 4.3.1.13 DiagReturnFlags

DiagReturnFlags specify what type of data should be returned when querying specific diag data packets. For more information on what they provide, see [DiagPacket](#).

```
enum DiagReturnFlags
{
    SESSION_INDEX           = 0x00000001,
    PROTOCOL_INDEX          = 0x00000002,
    RECEIVE_TIME_DATA       = 0x00000004,
    RECEIVE_TIME_STRING     = 0x00000008,
    PACKET_TYPE             = 0x00000010,
    PACKET_ID               = 0x00000020,
    PACKET_NAME             = 0x00000040,
    BINARY_PAYLOAD          = 0x00000080,
    PARSED_TEXT             = 0x00000100,
    TIME_STAMP_DATA         = 0x00000200,
    TIME_STAMP_STRING       = 0x00000400,
    SUBSCRIPTION_ID         = 0x00000800,
    PROCESSOR_ID            = 0x00001000,
    HW_TIME_STAMP_DATA      = 0x00002000,
    HW_TIME_STAMP_STRING    = 0x00004000,
    ULOG_SOURCE             = 0x00008000,
```

```

MORE_RESPONSES_FLAG = 0x00010000,
SUMMARY_TEXT        = 0x00020000,
QDSS_CHANNEL_ID     = 0x00040000,
QDSS_MASTER_ID      = 0x00080000,
QDSS_AT_ID          = 0x00100000,
DEFAULT_FORMAT_TEXT  = 0x00200000,
CALL_FRAME_NUMBER    = 0x00400000,
TIME_STAMP_TOD_ADJUSTED_DATA = 0x00800000
TIME_STAMP_TOD_ADJUSTED_STRING = 0x01000000
PACKET_SIZE          = 0x02000000
}

```

#### 4.3.1.14 DiagReturns

The DiagReturns type specifies information to be queried regarding a DiagPacket. The data specified is returned in the resulting [DiagPacket](#).

```

struct DiagReturns
{
    1: optional DiagReturnFlags flags;
    2: optional list<string> queries;
}

```

##### Fields

Flags	What fields to return in the resulting <a href="#">DiagPacket</a> items. Flags specified at this value apply only to the specific item specified in the fieldQueries of <a href="#">DiagReturnConfig</a> . Flags put in the <a href="#">DiagReturnConfig</a> apply to all packets.
Queries	Specifies queries to make for field values in the log packet. For information on queries, see <a href="#">Field</a> queries.

#### 4.3.1.15 DiagReturnConfig

DiagReturnConfig encompasses all return configuration information for a DiagPacket.

```

struct DiagReturnConfig
{
    1: DiagReturnFlags flags;
    2: optional map<DiagPacketType, map<string, DiagReturns> >
fieldQueries;
    3: optional bool diagTimeSorted;
}

```

##### Fields

Flags	What fields to return in the resulting <a href="#">DiagPacket</a> items. Flags specified in this structure apply to all resulting <a href="#">DiagPackets</a> . To specify fields per packet type or ID, use the flags in <a href="#">DiagReturns</a> , accessed using the fieldQueries field.
fieldQueries	Specialized returns for different packets based on ID. Each <a href="#">DiagPacketType</a> has a map of DiagReturns configuration keyed on the unique ID. The string value for

	the key of the secondary map can specify either an ID or the name of the item. For ID definitions, see <a href="#">DiagPacketType</a> .
diagTimeSorted	Allows specifying that a DataView should have the diag-based traffic sorted based on the target assigned diag time stamp. This option is only valid log session DataViews and only in postprocessing (not during live connections).

#### 4.3.1.16 DiagPacket

Return types for functions requesting a DiagPacket.

This structure contains many optional fields that can be filled using the [DiagReturnFlags](#) type. All fields other than `errorCode` are available only if [ErrorCode](#) is `DEVICE_NO_ERROR (0)`. Other fields are optionally available depending on the function and values specified in [DiagReturnFlags](#).

```
struct DiagPacket
{
    1: optional ErrorCode errorCode;
    2: optional DiagPacketType packetType;
    3: optional string packetId;
    4: optional bool moreResponsesFlag;
    5: optional i64 sessionId;
    6: optional i64 protocolIndex;
    7: optional string packetName;
    8: optional i64 timeStampData;
    9: optional string timeStampString;
    10: optional binary binaryPayload;
    11: optional string parsedText;
    12: optional i32 subscriptionId;
    13: optional i16 processorId;
    14: optional i64 hwTimeStampData;
    15: optional string hwTimeStampString;
    16: optional string ulogSource;
    17: optional i64 receiveTimeData;
    18: optional string receiveTimeString;
    19: optional string queryResultJson;
    20: optional string summaryText;
    21: optional i64 transactionId;
    22: optional i16 qdssChannelId;
    23: optional i16 qdssMasterId;
    24: optional i8 qdssAtid;
    25: optional string defaultFormatText;
    26: optional i16 callFrameNumber;
    27: optional i64 timeStampTodAdjustedData;
    28: optional string timeStampTodAdjustedString;
    29: optional list<string> formatStringArguments;
    30: optional i16 packetSize;
}
```

## Fields

errorCode	Code if an error occurred. All other fields available only when <code>DEVICE_NO_ERROR</code> (0) is set in <a href="#">ErrorCode</a> .
packetType	Diagnostic packet data type (defined in <a href="#">DiagPacketType</a> ). Values: <ul style="list-style-type: none"> <li>LOG_PACKET – Log ID (hex)</li> <li>EVENT – Event ID</li> <li>NV_ITEM – NV ID</li> <li>DEBUG_MSG – Subsystem ID/level</li> <li>REQUEST – DIAG command code</li> <li>RESPONSE – DIAG command code</li> <li>SUBSYS_REQUEST – Subsystem ID/subsystem command</li> <li>SUBSYS_RESPONSE – Subsystem ID/subsystem command</li> <li>SUBSYSV2_REQUEST – Subsystem ID/subsystem command</li> <li>SUBSYSV2_IMMEDIATE_RESPONSE – Subsystem ID/subsystem command</li> <li>SUBSYSV2_DELAYED_RESPONSE – Subsystem ID/subsystem command</li> <li>QTRACE – Client ID/buffer ID</li> <li>QSH_METRIC – Client ID/metric ID</li> </ul>
packetId	Packet ID based on type.
moreResponsesFlag	TRUE only for subsystem version 2 responses that anticipate further delayed responses from the corresponding request.
sessionIndex	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
protocolIndex	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
packetName	Description of the packet based on the <code>type</code> and <code>id</code> combination. Available when <code>PACKET_NAME</code> is set in <a href="#">DiagReturnFlags</a> .
timeStampData	Raw data for the DIAG-assigned timestamp of the packet. Timestamps are set by DIAG for LOG_PACKET, EVENT, DEBUG_MSG, QTRACE, and QSH_METRIC types. All others are interpolated by QUTS. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when <code>TIME_STAMP_DATA</code> is set in <a href="#">DiagReturnFlags</a> .
timeStampString	String formatted DIAG-assigned timestamp of the packet. Timestamps are set by DIAG for LOG_PACKET, EVENT, DEBUG_MSG, QTRACE, and QSH_METRIC types. All others are interpolated by QUTS. Available when <code>TIME_STAMP_STRING</code> is set in <a href="#">DiagReturnFlags</a> .
binaryPayload	Raw binary payload of the response starting at the command code. Available when <code>BINARY_PAYLOAD</code> is set in <a href="#">DiagReturnFlags</a> .

parsedText	Parsed text of the payload. PARSED_TEXT turns on the flag which returns formatted full parsed text for the log packet, ( i.e bUseFormat is set to true). Available when PARSED_TEXT is set in <a href="#">DiagReturnFlags</a> .
subscriptionId	ID of the subscription for the SIM from which the response came. Not all packets have a Subscription ID. If no ID of the subscription information is available for the packet, this field is set to -1. Available when SUBSCRIPTION_ID is set in <a href="#">DiagReturnFlags</a> .
processorId	Processor ID of the processor from which the response came. Not all packets have a processor ID. If no processor ID information is available for the packet, this field is set to 0xFF. Available when PROCESSOR_ID is set in <a href="#">DiagReturnFlags</a> .
hwTimeStampData	Raw data for the QDSS-assigned hardware timestamp. Packets not received from QDSS are assigned a timestamp of 0. Timestamp format is 100 ns ticks. Available when HW_TIME_STAMP_DATA is set in <a href="#">DiagReturnFlags</a> .
hwTimeStampString	String-formatted QDSS-assigned hardware timestamp. Packets not received from QDSS have an empty string. Available when HW_TIME_STAMP_STRING is set in <a href="#">DiagReturnFlags</a> .
uLogSource	Source name for the ULog-originated packet. Not all packets have a ULog source name. Packets with no source have an empty string. Available when ULOG_SOURCE is set in <a href="#">DiagReturnFlags</a> .
receiveTimeData	Raw data for the time when the packet was received by QUTS. Receive timestamps are consistent across all connections in QUTS and can be used for rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when RECEIVE_TIME_DATA is set in <a href="#">DiagReturnFlags</a> .
receiveTimeString	String formatted time when the packet was received by QUTS. Available when RECEIVE_TIME_STRING is set in <a href="#">DiagReturnFlags</a> .
queryResultJson	Returns the results of any queries specified in DiagReturnConfig (see <a href="#">DiagReturnConfig</a> ) in a single JSON formatted string.
summaryText	Single line summary text for a packet based on its type. This is similar to the QXDM Professional™ summary text. For debug messages, it will be the formatted output of the debug message, including file and line number. For events, it will be a single line summary of the payload. For RRC messages it will contain information about the signaling message.
transactionId	The transaction id received when the packet was sent.
qdssChannelId	QDSS data channel id.
qdssMasterId	QDSS data master id.

qdssAtid	QDSS data AT ID.
defaultFormatText	Return parsed text in default format. DEFAULT_FORMAT_TEXT turns off the flag which returns formatted full parsed text for the log packet, ( i.e bUseFormat is set to false). Available when DEFAULT_FORMAT_TEXT is set in <a href="#">DiagReturnFlags</a>
callFrameNumber	Available when CALL_FRAME_NUMBER is set in <a href="#">DiagReturnFlags</a>
timestampTodAdjustedData	Available when TIME_STAMP_TOD_ADJUSTED_DATA is set in <a href="#">DiagReturnFlags</a>
timestampTodAdjustedString	Available when TIME_STAMP_TOD_ADJUSTED_STRING is set in <a href="#">DiagReturnFlags</a>
formatStringArguments	Formatted string arguments
packetSize	Return packet size in bytes. Available when PACKET_SIZE is set in <a href="#">DiagReturnFlags</a>

#### 4.3.1.17 QmiPacketType

The QmiPacketType enum indicates whether a data packet is a request, response, or asynchronous indication.

```
enum QmiPacketType
{
    QMI_REQUEST = 0,
    QMI_RESPONSE,
    QMI_INDICATION
}
```

#### 4.3.1.18 QmiPacketFilter

The QmiPacketFilter specifies a list of message IDs to include.

```
struct QmiPacketFilter
{
    1: optional map<QmiPacketType, list<string> > idOrNameMask;
}
```

##### Fields

idOrNameMask	A list that can include a number of IDs. Each can be either the numerical message ID or the message name.
--------------	---

#### 4.3.1.19 QmiReturnFlags

Flags for requesting the types of data to return when calling functions that return a [QmiPacket](#).

```
enum QmiReturnFlags
{
    SESSION_INDEX = 0x00000001,
```

```

    PROTOCOL_INDEX      = 0x00000002,
    RECEIVE_TIME_DATA   = 0x00000004,
    RECEIVE_TIME_STRING = 0x00000008,
    PACKET_TYPE         = 0x00000010,
    PACKET_NAME         = 0x00000040,
    BINARY_PAYLOAD      = 0x00000080,
    PARSED_XML          = 0x00000100,
    SERVICE_ID          = 0x00000200,
    MESSAGE_ID          = 0x00000400
}

```

#### 4.3.1.20 QmiReturns

The QmiReturns type specifies return information for a single item, as specified in [QmiReturnConfig](#).

```

struct QmiReturns
{
    1: optional QmiReturnFlags flags;
    2: optional list<string> queries;
}

```

##### Fields

Flags	What fields to return in resulting <a href="#">QmiPacket</a> . Flags specified at this value apply only to the specific item specified in the queries of <a href="#">QmiReturnConfig</a> . Flags put in the <a href="#">QmiReturnConfig</a> object apply to all packets.
Queries	Specifies queries to make for field values in the log packet. See <a href="#">Field</a> queries.

#### 4.3.1.21 QmiReturnConfig

The QmiReturnConfig type specifies return information for all resulting [QmiPacket](#).

```

struct QmiReturnConfig
{
    1: QmiReturnFlags flags;
    2: optional map<QmiPacketType, map<string, QmiReturns> >
    fieldQueries;
}

```

##### Fields

Flags	What fields to return in resulting <a href="#">QmiPacket</a> . Flags specified in this structure apply to all resulting <a href="#">QmiPackets</a> . To specify fields per packet type or ID, use the flags in <a href="#">QmiReturns</a> , accessed using the queries field.
Queries	Specialized returns for different packets based on ID. Each <a href="#">QmiPacketType</a> has a map of <a href="#">QmiReturns</a> configuration keyed on the unique ID. The string value for the key of the secondary map can specify either an ID or the name of the message.



#### 4.3.1.22 QmiPacket

Return type for functions requesting a QMI packet.

This structure contains many optional fields that can be filled using [QmiReturnFlags](#). All fields other than `errorCode` are available only if `ErrorCode` is `DEVICE_NO_ERROR (0)`.

Other fields are optionally available depending on the function and the values specified in [QmiReturnFlags](#).

```
struct QmiPacket
{
    1: optional ErrorCode errorCode;
    2: optional QmiPacketType packetType;
    3: optional i32 serviceId;
    4: optional i16 messageId;
    5: optional i64 sessionIndex;
    6: optional i64 protocolIndex;
    7: optional string packetName;
    8: optional binary binaryPayload;
    9: optional string parsedXml;
    10: optional i64 receiveTimeData;
    11: optional string receiveTimeString;
    12: optional string queryResultJson;
    13: optional i64 transactionId;
}
```

#### Fields

<code>errorCode</code>	Code if an error occurred. All other fields are available only when <code>DEVICE_NO_ERROR (0)</code> is set in <a href="#">ErrorCode</a> .
<code>packetType</code>	QMI packet data type according to <a href="#">QmiPacketType</a> .
<code>serviced</code>	ID of the QMI service this connection has opened.
<code>messageId</code>	ID of the QMI message packet.
<code>sessionIndex</code>	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
<code>protocolIndex</code>	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
<code>packetName</code>	Description of the packet based on the <code>serviceId</code> and <code>messageId</code> combination. Available when <code>PACKET_NAME</code> is set in <a href="#">QmiReturnFlags</a> .
<code>binaryPayload</code>	Raw binary payload of the response starting at the command code. Available when <code>BINARY_PAYLOAD</code> is set in <a href="#">QmiReturnFlags</a> .
<code>parsedXml</code>	Parsed XML string of the payload. Available when <code>PARSED_XML</code> is set in <a href="#">QmiReturnFlags</a> .
<code>receiveTimeData</code>	Raw data for the time when the packet was received by QUTS.

	Receive timestamps are consistent across all connections in QUTS and can be used for a rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when RECEIVE_TIME_DATA is set in <a href="#">QmiReturnFlags</a> .
receiveTimeString	String formatted time when the packet was received by QUTS. Available when RECEIVE_TIME_STRING is set in <a href="#">QmiReturnFlags</a> .
queryResultJson	Returns the results of any queries specified in QmiReturnConfig (see <a href="#">QmiReturnConfig</a> ) in a single JSON formatted string.
transactionId	The transaction id received when the packet was sent.

4.3.1.23 AdbPacketFilter

The AdbPacketFilter provides an optional filter for determining ADB responses to collect.

```
struct AdbPacketFilter
{
    1: optional string regexFilter;
}
```

Fields

regexFilter	Regular expression to filter based on the ADB response
-------------	--

4.3.1.24 AdbReturnFlags

The AdbReturnFlags type determines what information to return for [AdbPackets](#).

```
enum AdbReturnFlags
{
    SESSION_INDEX      = 0x00000001,
    PROTOCOL_INDEX     = 0x00000002,
    RECEIVE_TIME_DATA  = 0x00000004,
    RECEIVE_TIME_STRING = 0x00000008,
    PACKET_TEXT        = 0x00000100
}
```

4.3.1.25 AdbReturnConfig

The AdbReturnConfig specifies what type of information to return for each [AdbPacket](#).

```
struct AdbReturnConfig
{
    1: AdbReturnFlags flags;
}
```

## Fields

Flags	Determines what information to return. See <a href="#">AdbReturnFlags</a>
-------	---

### 4.3.1.26 AdbPacket

Return type for functions requesting a ADB packet.

This structure contains many optional fields that can be filled using the [AdbReturnFlags](#). All fields other than `errorCode` are available only if `ErrorCode` is `DEVICE_NO_ERROR (0)`. Other fields are optionally available depending on the function and the values specified in [AdbReturnFlags](#).

```
struct AdbPacket
{
    1: optional ErrorCode errorCode;
    2: optional i64 sessionIndex;
    3: optional i64 protocolIndex;
    4: optional string packetText;
    5: optional i64 receiveTimeData;
    6: optional string receiveTimeString;
}
```

## Fields

<code>errorCode</code>	Code if an error occurred. All other fields are available only when <code>DEVICE_NO_ERROR (0)</code> is set in <a href="#">ErrorCode</a> .
<code>sessionIndex</code>	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
<code>protocolIndex</code>	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
<code>packetText</code>	The ADB text of the packet
<code>receiveTimeData</code>	Raw data for the time when the packet was received by QUTS. Receive timestamps are consistent across all connections in QUTS and can be used for a rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when <code>RECEIVE_TIME_DATA</code> is set in <a href="#">AdbReturnConfigQmiReturnFlags</a> .
<code>receiveTimeString</code>	String formatted time when the packet was received by QUTS. Available when <code>RECEIVE_TIME_STRING</code> is set in <a href="#">AdbReturnConfig</a> .

4.3.1.27 SaharaPacketFilter

SaharaPacketFilter provides an optional filter for determining Sahara packets to collect.

```
struct SaharaPacketFilter
{
    1: optional list<string> idOrNameMask;
}
```

Fields

idOrNameMask	List of Sahara packet IDs or packet names to include.
--------------	---

#### 4.3.1.28 SaharaReturnFlags

The SaharaReturnFlags type determines what information to return for [SaharaPackets](#).

```
enum SaharaReturnFlags
{
    SESSION_INDEX      = 0x00000001,
    PROTOCOL_INDEX     = 0x00000002,
    RECEIVE_TIME_DATA  = 0x00000004,
    RECEIVE_TIME_STRING = 0x00000008,
    PACKET_ID          = 0x00000020,
    PACKET_NAME        = 0x00000040,
    BINARY_PAYLOAD     = 0x00000080
}
```

#### 4.3.1.29 SaharaReturnConfig

The SaharaReturnConfig specifies what type of information to return for each [SaharaPacket](#).

```
struct SaharaReturnConfig
{
    1: SaharaReturnFlags flags;
}
```

##### Fields

Flags	Determines what information to return. See <a href="#">SaharaReturnFlags</a> .
-------	--

#### 4.3.1.30 SaharaPacket

Return type for functions requesting a Sahara packet.

This structure contains many optional fields that can be filled using the SaharaReturnFlags (see [SaharaReturnFlags](#)). All fields other than `errorCode` are available only if `ErrorCode` is `DEVICE_NO_ERROR (0)`. Other fields are optionally available depending on the function and the values specified in SaharaReturnFlags.

```
struct SaharaPacket
{
    1: optional ErrorCode errorCode;
    2: optional i32 packetId;
    3: optional i64 sessionId;
    4: optional i64 protocolIndex;
    5: optional string packetName;
    6: optional binary binaryPayload;
    7: optional i64 receiveTimeData;
    8: optional string receiveTimeString;
}
Fields
```

errorCode	Code if an error occurred. All other fields are available only when <code>DEVICE_NO_ERROR (0)</code> is set in <a href="#">ErrorCode</a> .
packetId	Sahara packet ID.
sessionIndex	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
protocolIndex	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
packetName	The name of the Sahara packet.
binaryPayload	The binary payload of the Sahara packet.
receiveTimeData	Raw data for the time when the packet was received by QUTS. Receive timestamps are consistent across all connections in QUTS and can be used for a rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when <code>RECEIVE_TIME_DATA</code> is set in <a href="#">QmiReturnFlags</a> .
receiveTimeString	String formatted time when the packet was received by QUTS. Available when <code>RECEIVE_TIME_STRING</code> is set in <a href="#">QmiReturnFlags</a> .

#### 4.3.1.31 FastbootPacketFilter

The FastbootPacketFilter provides an optional filter for determining what Fastboot messages to collect.

```
struct FastbootPacketFilter
{
    1: optional string regexFilter;
}
```

#### Fields

regexFilter	Regular expression to filter based on the Fastboot message
-------------	--

#### 4.3.1.32 FastbootReturnFlags

The FastbootReturnFlags type determines what information to return for [FastbootPackets](#).

```
enum FastbootReturnFlags
{
    SESSION_INDEX          = 0x00000001,
    PROTOCOL_INDEX         = 0x00000002,
    RECEIVE_TIME_DATA      = 0x00000004,
    RECEIVE_TIME_STRING     = 0x00000008,
    PACKET_TEXT            = 0x00000100
}
```

#### 4.3.1.33 FastbootReturnConfig

The FastbootReturnConfig specifies what type of information to return for each [FastbootPacket](#).

```
struct FastbootReturnConfig
{
    1: FastbootReturnFlags flags;
}
```

##### Fields

flags	Determines what information to return. See <a href="#">FastbootPacketFilter</a> .
-------	---

#### 4.3.1.34 FastbootPacket

Return type for functions requesting a Fastboot packet.

This structure contains many optional fields that can be filled using the FastbootReturnFlags (see [FastbootPacketFilter](#)). All fields other than `errorCode` are available only if `ErrorCode` is `DEVICE_NO_ERROR (0)`. Other fields are optionally available depending on the function and the values specified in [FastbootReturnFlags](#).

```
struct FastbootPacket
{
    1: optional ErrorCode errorCode;
    2: optional i64 sessionIndex;
    3: optional i64 protocolIndex;
    4: optional string packetText;
    5: optional i64 receiveTimeData;
    6: optional string receiveTimeString;
}
```

## Fields

errorCode	Code if an error occurred. All other fields are available only when <code>DEVICE_NO_ERROR (0)</code> is set in <a href="#">ErrorCode</a> .
sessionIndex	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
protocolIndex	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
packetText	The Fastboot text of the packet
receiveTimeData	Raw data for the time when the packet was received by QUTS. Receive timestamps are consistent across all connections in QUTS and can be used for a rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when <code>RECEIVE_TIME_DATA</code> is set in <a href="#">FastbootReturnConfig</a> .
receiveTimeString	String formatted time when the packet was received by QUTS. Available when <code>RECEIVE_TIME_STRING</code> is set in <a href="#">FastbootReturnConfig</a> .

### 4.3.1.35 AnnotationPacketFilter

The AnnotationPacketFilter provides an optional filter for determining what annotation messages to collect.

```
struct AnnotationPacketFilter
{
    1: optional bool includeAnnotations;
    2: optional list<i64> messageIdFilter;
}
```

## Fields

includeAnnotations	Boolean flag to include (true) or exclude (false) annotation messages
messageIdFilter	List of message IDs to include. If not set and includeAnnotations is true, all annotation packets are included.



### 4.3.1.36 AnnotationPacket

Return type for functions requesting an annotation packet.

```
struct AnnotationPacket
{
    1: optional ErrorCode errorCode;
    2: optional i64 sessionIndex;
    3: optional i64 protocolIndex;
    4: optional string parsedText;
    5: optional i64 messageId;
    6: optional i64 receiveTimeData;
    7: optional string receiveTimeString;
}
```

#### Fields

errorCode	Code if an error occurred.
sessionIndex	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
protocolIndex	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
parsedText	The annotation text of the packet
messageId	The message id of the packet
receiveTimeData	Raw data for the time when the packet was received by QUTS. Receive timestamps are consistent across all connections in QUTS and can be used for a rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when RECEIVE_TIME_DATA is set in <a href="#">FastbootReturnConfig</a> .
receiveTimeString	String formatted time when the packet was received by QUTS. Available when RECEIVE_TIME_STRING is set in <a href="#">FastbootReturnConfig</a> .

### 4.3.1.37 BuildInfo

Structure that contains all relevant information for a specific protocol.

```
struct BuildInfo{
    1: optional string msmRevision;
    2: optional string mobileModelId;
    3: optional string mobileSoftwareRevision;
    4: optional string mobileModelName;
}
```

## Fields

msmRevision	<p>An extension of the MSM_VER field from the version number response packet. Length and format is dependent on the Version field. Values are:</p> <ul style="list-style-type: none"> <li>■ 0 – Length is 16 bits and derived from the 16-bit hardware version register. Format varies depending on the target build.</li> <li>■ 1 – Length is 20 bits and derived from the 32-bit hardware version register. Format is described by the following offsets in the 32-bit hardware version register: <ul style="list-style-type: none"> <li>▫ Bits 19:4 = Bits 27:12 of the hardware version register</li> <li>▫ Bits 3:0 = Bits 31:28 of the hardware version register</li> </ul> </li> <li>■ 2 – Length is 32 bits and is the value of the full 32-bit hardware version register</li> </ul>
mobileModelId	Manufacturer's mobile model number
mobileSoftwareRevision	Mobile software revision as a string
mobileModelName	Mobile model as a string

### 4.3.1.38 NvItem

Structure that contains all NV metadata for its definition.

```
struct NvItem
{
    1: optional i64 id;
    2: optional string name;
    3: optional string description;
    4: optional string category;
}
```

## Fields

id	The numeric assignment of the NV Item as seen in QXDM or PCAT
name	The description of the NV item or EFS path
Description	A text description about the NV Item
category	Category the NV item falls into

#### 4.3.1.39 RxTxInfo

Contains all throughput information for a specific connection.

```
struct RxTxInfo
{
    1: i64 rxTotalPacketCount;
    2: i64 txTotalPacketCount;
    3: i64 rxTotalByteCount;
    4: i64 txTotalByteCount;
    5: double rxInstThroughput;
    6: double txInstThroughput;
}
```

#### Fields

rxTotalPacketCount	Total number of packets that have been received on this protocol since it was opened
txTotalPacketCount	Total number of packets that have been sent on this protocol since it was opened
rxTotalByteCount	Total number of bytes that have been received on this protocol since it was opened
txTotalByteCount	Total number of bytes that have been sent on this protocol since it was opened
rxInstThroughput	Instantaneous receive throughput. This is calculated as the number of bytes received based on the previous half second.
txInstThroughput	Instantaneous transmit throughput. This is calculated as the number of bytes sent based on the previous half second.

**NOTE:** All throughput stats are combined across any clients that may be using a given protocol. This means that the time the connection was opened may have been before the current client started using it. It also means throughput can be affected based on the usage of other active QUTS clients.

#### 4.3.1.40 QShrink4DownloadState

Used to specify the current state qshrink is in using getQShrinkState.

```
enum QShrink4DownloadState
{
    QSR4_STATE_INIT = 0,
    QSR4_STATE_DOWNLOADING,
    QSR4_STATE_DOWNLOADED,
    QSR4_STATE_LOADED,
    QSR4_STATE_FAILED
};
```

4.3.1.41 NmeaPacketFilter

The NmeaPacketFilter provides an optional filter for determining what Nmea messages to collect.

```
struct NmeaPacketFilter
{
    1: optional list<string> nameMask;
}
```

Fields

nameMask	List of mask strings. Possible values for individual mask strings in the list are "\$GPGGA" and "\$GPRMC"
----------	---

4.3.1.42 NmeaReturnFlags

The NmeaReturnFlags type determines what information to return for [NmeaPackets](#).

```
enum NmeaReturnFlags
{
    SESSION_INDEX          = 0x00000001,
    PROTOCOL_INDEX         = 0x00000002,
    RECEIVE_TIME_DATA      = 0x00000004,
    RECEIVE_TIME_STRING    = 0x00000008,
    PACKET_ID              = 0x00000020,
    PACKET_TEXT            = 0x00000100
}
```

4.3.1.43 NmeaReturnConfig

The NmeaReturnConfig specifies what type of information to return for each [NmeaPacket](#).

```
struct NmeaReturnConfig
{
    1: NmeaReturnFlags flags;
}
```

Fields

flags	Determines what information to return. See <a href="#">NmeaReturnFlags..</a>
-------	--

#### 4.3.1.44 NmeaPacket

Return type for functions requesting a Nmea packet.

This structure contains many optional fields that can be filled using the [NmeaReturnFlags](#). All fields other than `errorCode` are available only if `ErrorCode` is `DEVICE_NO_ERROR (0)`. Other fields are optionally available depending on the function and the values specified in [NmeaReturnFlags](#).

```
struct NmeaPacket
{
    1: optional ErrorCode errorCode;
    2: optional i64 sessionIndex;
    3: optional i64 protocolIndex;
    4: optional string packetText;
    5: optional i64 receiveTimeData;
    6: optional string receiveTimeString;
    7: optional string packetId;
}
```

#### Fields

<code>errorCode</code>	Code if an error occurred. All other fields are available only when <code>DEVICE_NO_ERROR (0)</code> is set in <a href="#">ErrorCode</a> .
<code>sessionIndex</code>	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
<code>protocolIndex</code>	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
<code>packetText</code>	The Nmea text of the packet
<code>receiveTimeData</code>	Raw data for the time when the packet was received by QUTS. Receive timestamps are consistent across all connections in QUTS and can be used for a rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when <code>RECEIVE_TIME_DATA</code> is set in <a href="#">NmeaReturnConfig</a> .
<code>receiveTimeString</code>	String formatted time when the packet was received by QUTS. Available when <code>RECEIVE_TIME_STRING</code> is set in <a href="#">NmeaReturnConfig</a> .

#### 4.3.1.45 SpcStatus

The SpcStatus specifies the return type for a checkSpc API call.

```
struct SpcStatus
{
    1: Common.ErrorType errorType;
    2: bool status;
}
```

#### 4.3.1.46 OperatingMode

OperatingMode specifies the various possible operating modes that can be set for [setOperatingMode](#) or received from [getOperatingMode](#). Only some can be used as [setOperatingMode\(\)](#) input parameters. Please refer to the comments next to enum to know if they can be used in conjunction with [setOperatingMode\(\)](#).

```
enum OperatingMode
{
    MODE_NONE = -1,
    OFFLINE_ANALOG = 0,
    OFFLINE_DIGITAL = 1,
    OFFLINE_FACTORY_TEST = 3,
    ONLINE = 4,
    LOW_POWER = 5,
    POWER_OFF = 6,
    OFFLINE,          // not valid for setOperatingMode() API
    RESET,            // not valid for setOperatingMode() API
    NET_TEST_GW,      // not valid for setOperatingMode() API
    OFFLINE_IF_NOT_FTM,
    ONLINE_TX_DISABLED
}
```

#### 4.3.1.47 OperatingModeInfo

OperatingModeInfo specifies the return type for the getOperatingMode API call.

```
struct OperatingModeInfo
{
    1: Common.ErrorType errorType;
    2: OperatingMode operatingMode;
}
```

#### 4.3.1.48 ProtocolState

The protocols current state.

```
enum ProtocolState
{
    STATE_AVAILABLE = 0,
    STATE_DISCONNECTED = 1,
}
```

```
STATE_UNRESPONSIVE = 2,  
STATE_INITIALIZING = 3  
}
```

The state of the protocol is determined based on the specific protocol type. The following table describes the various protocol states.

STATE_AVAILABLE	Indicates that the protocol is connected and responding as expected. Not all protocols have an availability check; for protocols not mentioned or for unknown protocols, the state is set to STATE_AVAILABLE but QUTS does not validate the responsiveness of the protocol.  For diag, this state indicates it is successfully responding to loopback commands.  For QMI, this state indicates the driver has marked the protocol as ready on the device.
STATE_DISCONNECTED	Indicates the protocol has been successfully removed from the host. For USB, this means it would no longer show up in the device manager. For TCP this means the connection was removed by the user.
STATE_UNRESPONSIVE	Indicates that the protocol has been connected but that the communication layer has stopped receiving data from the protocol. For example, for USB, the drivers are no longer able to communicate to the protocol, or for TCP, the connection has been lost.
STATE_INITIALIZING	Indicates the protocol has been discovered by the system but is not necessarily ready for use. If a protocol has no specific ready check, it immediately moves to STATE_AVAILABLE.  Protocols can be connected to while in STATE_INITIALIZING but the behavior will be undefined. If the use case does not expect the protocol to behave normally (for instance diag does not respond to status requests), the user may need to use it while in STATE_INITIALIZING.

#### 4.3.1.49 Direction

The direction of data transmission.

```
enum Direction  
{  
    DIR_RX = 1,  
    DIR_TX = 2  
}
```

#### 4.3.1.50 FlowControlStatus

The protocol flow control status on a direction.

```
enum FlowControlStatus  
{  
    FLOW_CONTROL_OFF = 0,  
    FLOW_CONTROL_ON = 1  
}
```

#### 4.3.1.51 LockState

The lock state of a protocol.

```
enum LockState
{
    LOCK_STATE_OFF = 0,
    LOCK_STATE_ON = 1
}
```

#### 4.3.1.52 LockStatus

The protocol lock status containing detailed lock information.

```
struct LockStatus
{
    1: LockState lockState;
    2: optional i64 clientId;
    3: optional string reason;
}
```

#### 4.3.1.53 GpsPacket

GPS packet information.

```
struct GpsPacket
{
    1: optional ErrorCode errorCode;
    2: optional string latitude;
    3: optional string longitude;
    4: optional string altitude;
    5: optional string speed;
    6: optional string time;
    7: optional string date;
}
```



4.3.1.54 AdplReturnFlags

The AdplReturnFlags type determines what information to return for [AdplPackets](#). Use Raw Service to enable DPL/ADPL logging.

```
enum AdplReturnFlags
{
    SESSION_INDEX          = 0x00000001,
    PROTOCOL_INDEX         = 0x00000002,
    RECEIVE_TIME_DATA      = 0x00000004,
    RECEIVE_TIME_STRING     = 0x00000008,
    PACKET_VERSION         = 0x00000010,
    BINARY_PAYLOAD         = 0x00000080
}
```

4.3.1.55 AdplReturnConfig

AdplReturnConfig specifies what type of information to return for each [AdplPacket](#). Use Raw Service to enable DPL /ADPL logging.

```
struct AdplReturnConfig
{
    1: AdplReturnFlags flags;
}
```

Fields

Flags	Determines what information to return. See <a href="#">AdplReturnFlags</a> .
-------	--

4.3.1.56 AdplPacket

Return type for functions requesting an ADPL packet. Use Raw Service to enable DPL / ADPL logging.

```
struct AdplPacket
{
    1: optional ErrorCode errorCode;
    2: optional i64 sessionIndex;
    3: optional i64 protocolIndex;
    4: optional i8 version;
    5: optional string binaryPayload;
    6: optional i64 receiveTimeData;
    7: optional string receiveTimeString;
}
```

#### 4.3.1.57 QdssPacketFilter

QdssPacketFilter provides an optional filter for determining what QDSS messages to collect.

```
struct QdssPacketFilter
{
    1: list<i8> atIds;
    2: list<i16> stpv2Atids;
    3: list<i16> stmAtids;
    4: optional list<i16> diagEntityId;
}
```

##### Fields

atIds	List of AT ID to stream
stpv2Atids	List of AT ID which are of type stpv2
stmAtids	List of AT ID which are of type stm
diagEntityId	List of entities which represent diag data

#### 4.3.1.58 QdssReturnFlags

The QdssReturnFlags type determines what information to return for [QdssPackets](#).

```
enum QdssReturnFlags
{
    SESSION_INDEX          = 0x00000001,
    PROTOCOL_INDEX         = 0x00000002,
    RECEIVE_TIME_DATA      = 0x00000004,
    RECEIVE_TIME_STRING    = 0x00000008,
    AT_ID                  = 0x00000020,
    BINARY_PAYLOAD         = 0x00000080,
    HW_TIME_STAMP_DATA     = 0x00002000,
    HW_TIME_STAMP_STRING   = 0x00004000,
    CHANNEL_ID             = 0x00020000,
    MASTER_ID              = 0x00040000,
}
```

#### 4.3.1.59 QdssReturnConfig

QdssReturnConfig specifies what type of information to return for each [QdssPacket](#).

```
struct QdssReturnConfig
{
    1: QdssReturnFlags flags;
}
```

## Fields

flags	Determines what information to return. See <a href="#">QdssReturnFlags</a>
-------	--

### 4.3.1.60 QdssPacket

Return type for functions requesting a QDSS packet.

```
struct QdssPacket
{
    1: optional ErrorCode errorCode;
    2: optional i64 sessionIndex;
    3: optional i64 protocolIndex;
    4: optional i64 receiveTimeData;
    5: optional string receiveTimeString;
    6: optional i8 atid;
    7: optional binary binaryPayload;
    8: optional i64 hwTimeStampData;
    9: optional string hwTimeStampString;
    10: optional i16 channelId;
    11: optional i16 masterId;
}
```

## Fields

errorCode	Code if an error occurred.
sessionIndex	Session index of the packet. The index is relative to incoming packets from all active protocols in QUTS.
protocolIndex	Protocol index of the packet. The index is relative to incoming packets from only this instance of the protocol in QUTS.
receiveTimeData	Raw data for the time when the packet was received by QUTS. Receive timestamps are consistent across all connections in QUTS and can be used for rudimentary sorting across protocols. However, depending on the protocol implementations and transport layers sent from the device, receive times might not be consistent with relative packet creation/send times. Timestamp format is Windows FILETIME, 100 ns ticks from midnight, Jan 1 1600 UTC. Available when <code>RECEIVE_TIME_DATA</code> is set in <a href="#">QdssReturnFlags</a> .
receiveTimeString	String formatted time when the packet was received by QUTS. Available when <code>RECEIVE_TIME_STRING</code> is set in <a href="#">QdssReturnFlags</a> .
atid	QDSS AT ID value of the packet.
binaryPayload	Raw binary payload of the response starting at the command code. Available when <code>BINARY_PAYLOAD</code> is set in <a href="#">QdssReturnFlags</a> .

hwTimeStampData	Raw data for the QDSS-assigned hardware timestamp. Packets not received from QDSS are assigned a timestamp of 0. Timestamp format is 100 ns ticks. Available when HW_TIME_STAMP_DATA is set in <a href="#">QdssReturnFlags</a> .
hwTimeStampString	String-formatted QDSS-assigned hardware timestamp. Packets not received from QDSS have an empty string. Available when HW_TIME_STAMP_STRING is set in <a href="#">QdssReturnFlags</a> .
channelId;	QDSS data channel ID.
masterId;	QDSS data master ID.

#### 4.3.1.61 QmiConnectionOptions

Specifies connection options for qmiconnection.

```
struct QmiConnectionOptions
{
    1: optional i64 protocolHandle;
}
```

#### 4.3.1.62 FunctionArea

The FunctionArea type specifies functional areas to be logged, used by [enableFunctionLog](#) and [disableFunctionLog](#) API.

```
enum FunctionArea
{
    FUNCTION_AREA_SAHARA_PORT_TRACE      = 0x00000001,
    FUNCTION_AREA_FIREHOSE_PORT_TRACE    = 0x00000002,
    FUNCTION_AREA_FIREHOSE_LOADER        = 0x00000003,
    FUNCTION_AREA_QDSS_DIAG_RAW_TRACE    = 0x00000004
}
```

#### 4.3.1.63 LogLevel

The LogLevel type specifies logging levels to be enabled in the log file.

```
enum LogLevel
{
    LOG_DEFAULT      = 0x00000000,
    LOG_DATA         = 0x00000001,
    LOG_DEBUG        = 0x00000002,
    LOG_INFO         = 0x00000004,
    LOG_WARNING      = 0x00000008,
    LOG_ERROR        = 0x00000010,
    LOG_EXCEPTION    = 0x00000020,
    LOG_FATAL        = 0x00000040,
    LOG_ALL          = 0x7FFFFFFF
}
```

#### 4.3.1.64 LogFormat

The LogFormat type specifies log file format.

```
enum LogFormat
{
    LOG_CSV      = 0x00000001,
    LOG_BINARY   = 0x00000002
}
```

#### 4.3.1.65 LogLayout

The LogLayout type specifies information that can be turned on in the log file for each log message.

```
enum LogLayout
{
    LOG_DATE           = 0,
    LOG_DATEUTC,
    LOG_DATETIME,
    LOG_DATETIMEUTC,
    LOG_TIME,
    LOG_TIMEUTC,
    LOG_TIMEZONE,
    LOG_EPOCH,
    LOG_UPTIME,
    LOG_RUNTIME,

    LOG_SEQUENCE,
    LOG_LOGSEQUENCE,
    LOG_LEVEL,
    LOG_MESSAGE,
    LOG_COMBINED,

    LOG_DATATYPE,
    LOG_DATALEN,
    LOG_DATA,

    LOG_LOGGER,
    LOG_CLASS,
    LOG_NAMESPACE,
    LOG_APPNAME,
    LOG_APPVER,
    LOG_PROCNAME,
    LOG_PID,
    LOG_PROCID,
    LOG_TID,
```

```
LOG_THREADID,
LOG_THREADNAME,
LOG_HOSTNAME,
LOG_USERNAME,

LOG_CALLERFILE,
LOG_CALLERPATH,
LOG_CALLERLINE,
LOG_CALLERMETHOD,
LOG_TRACE,

LOG_EXCEPTION,
LOG_EXCEPTIONMESSAGE,
LOG_EXCEPTIONNAME,
LOG_EXCEPTIONSTACK
}
```

4.3.1.66 LogOptions

Logging options to be used with logging APIs to customize log file properties.

```
struct LogOptions
{
    1: optional LogLevel    level;
    2: optional LogFormat   format;
    3: optional list<LogLayout> layout;
    4: optional i32         sizeRotationKB;
    5: optional string      savePath;
}
```

Fields

level	Logging level bitmask.
format	Log file format.
layout	List of LogLayout type to specify log file layout.
sizeRotationKB	Maximum log file size in KB before rotation.
savePath	Client specified path for saving log files.

4.3.1.67 CdmaProtocolRevision

Used to specify the version while setting the revision in [setCdmaProtocolRevision\(\)](#).

```
enum CdmaProtocolRevision
{
    IS_95A = 0,
    IS_95B = 4,
    IS_2000_Rev_0 = 6,
    IS_2000_Rev_A,
```

```

IS_2000_Rev_B,
IS_2000_Rev_C,
IS_2000_Rev_C2,
IS_2000_Rev_D
}

```

#### 4.3.1.68 WcdmaProtocolRevision

Used to specify the version while setting the revision in [setWcdmaProtocolRevision\(\)](#).

```

enum WcdmaProtocolRevision
{
    VB50 = 0x0B201303,
    VA70 = 0x0A201203,
    V9B0 = 0x09201207,
    VC80 = 0x0C201601,
    VAB1 = 0x0A201311,
    V930 = 0x09200912,
    V8A0 = 0x08201003,
    V860 = 0x08200903,
    V7G0 = 0x07201003,
    V790 = 0x07200805,
    V780 = 0x07200803,
    V770 = 0x07200712,
    V6F0 = 0x06200709,
    V6B0 = 0x06200609,
    V6A0 = 0x06200606,
    V690 = 0x06200603,
    V680 = 0x06200512,
    V590 = 0x05200406,
    V3I0 = 0x03200403,
    V3A0 = 0x03200203,
    V370 = 0x03200106,
    V350 = 0x03199900,
    VC81 = 0x0C201602
}

```

#### 4.3.1.69 BaudRate

Used to specify the baudRate in [configureProtocol\(\)](#). Not needed to be set for regular diag in general where quts uses default baudRate of 38400. For RUMI, set this to 115200. For NMEA, quts automatically sets baudrate to 4800 during override, so explicit set of baudrate is not needed for NMEA.

```

enum BaudRate
{
    BAUD_RATE_75 = 75,

```

```

BAUD_RATE_110    = 110,
BAUD_RATE_134    = 134,
BAUD_RATE_150    = 150,
BAUD_RATE_300    = 300,
BAUD_RATE_600    = 600,
BAUD_RATE_1200   = 1200,
BAUD_RATE_1800   = 1800,
BAUD_RATE_2400   = 2400,
BAUD_RATE_4800   = 4800,
BAUD_RATE_7200   = 7200,
BAUD_RATE_9600   = 9600,
BAUD_RATE_14400  = 14400,
BAUD_RATE_19200  = 19200,
BAUD_RATE_38400  = 38400,
BAUD_RATE_57600  = 57600,
BAUD_RATE_115200 = 115200,
BAUD_RATE_128000 = 128000
}

```

#### 4.3.1.70 ProtocolConfiguration

Used to configure protocol in [configureProtocol](#).

```

struct ProtocolConfiguration
{
    1: i64                protocolHandle;
    2: optional ProtocolType overrideProtocolType;
    3: optional BaudRate   baudRate;
    4: optional i64        sendTimeout;
}

```

#### Fields

protocolHandle	Protocol handle to configure. Mandatory field
overrideProtocolType	Optional. If unknown protocol needs to be overridden say e.g. unknown to NMEA or unknown to Diag (e.g. RUMI device), this parameter can be used to specify corresponding type.
baudRate	Baud rate to use. Not needed for NMEA where quts automatically sets 4800 baudrate for NMEA to work. For RUMI diag, use 115200.
sendTimeout (milliseconds)	Quts has default timeout of 2 seconds per packet, i.e. wait time for response before next packet is sent on Diag channel. For slower devices where responses are expected to come later than 2 seconds, this needs to be changed so Quts will wait for the corresponding time before considering as failed and sending next packet on diag channel. For RUMI, recommended to use 120000 (i.e. 120 seconds or 2 mins). Quts will however use max value of 300000 and min value of 1000 i.e. if values are specified outside these ranges, Quts will use the corresponding max or min value.



## 4.3.2 Functions

### 4.3.2.1 getLastError()

Queries for the error from the last function call.

#### Prototype

```
Common.ErrorType getLastError()
```

#### Returns

Error code and description.

### 4.3.2.2 getServicesList()

Gets a list of available services provided by QUTS.

#### Prototype

```
list<string> getServicesList()
```

#### Returns

List of available service names.

### 4.3.2.3 getDevicesForService()

Gets a list of devices that support a specified service.

#### Prototype

```
list<i64> getDevicesForService(1:string serviceName)
```

#### Parameters

serviceName	Service for which to find devices.
-------------	------------------------------------

#### Returns

List of devices.

### 4.3.2.4 getServicesForDevice()

Gets a list of services that are supported by a specified device.

#### Prototype

```
list<string> getServicesForDevice(1:i64 deviceHandle)
```

### Parameters

deviceHandle	Device for which to find services.
--------------	------------------------------------

### Returns

List of services.

#### 4.3.2.5 createService()

Creates a new instance of a specified service to be run on a specified device.

To make calls on the service's interface, a new RPC client must be set up on this service.

### Prototype

```
string createService(1:string serviceName,  
                    2:i64 deviceHandle)
```

### Parameters

serviceName	Service to create.
deviceHandle	Device on which to create the service.

### Returns

Unique identifier of the RPC service.

#### 4.3.2.6 getDeviceList()

Gets the information for all devices currently connected to QUTS.

### Prototype

```
list<Common.DeviceInfo> getDeviceList()
```

### Returns

List of available devices (see [DeviceInfo](#)).

Upon a failure, the list is empty (for the error code, call [getLastError\(\)](#)).

#### 4.3.2.7 getDeviceMode()

Returns current device mode of operations.

### Prototype

```
Common.DeviceMode getDeviceMode(1:i64 deviceHandle)
```

### Parameters

deviceHandle	Device on which to get the operation mode.
--------------	--

### Returns

Device operation mode bitmask (see [DeviceMode](#)).

#### 4.3.2.8 getProtocolList()

Gets a list of protocols available on a specified device.

### Prototype

```
list<Common.ProtocolInfo> getProtocolList(1:i64 deviceHandle)
```

### Parameters

deviceHandle	Device for which to get protocols.
--------------	------------------------------------

### Returns

List of available protocols (see [ProtocolInfo](#)).

Upon a failure, the list is empty (for the error code, call [getLastError\(\)](#)).

#### 4.3.2.9 overrideUnknownProtocol()

Allows an unknown protocol to be set and used as the specified protocol type.

### Prototype

```
Common.ErrorCode overrideUnknownProtocol(1:i64 protocolHandle,  
                                          2:Common.ProtocolType newType)
```

## Parameters

protocolHandle	Protocol to override.
newType	Type of protocol to use for this connection.

## Returns

None.

### 4.3.2.10 addTcpConnection()

Adds a TCP connection into QUTS. Once the TCP connection is created, it can be used the same as any other connection.

## Prototype

```
Common.ProtocolInfo addTcpConnection(1:i64 deviceHandle,  
                                     2:Common.ProtocolType protocolType,  
                                     3:bool bIsClient,  
                                     4:string description,  
                                     5:string host,  
                                     6:i64 port)
```

## Parameters

deviceHandle	The device to which to attach this connection. To create a new device, set to 0.
protocolType	What type of protocol this connection is (see <a href="#">AppException</a> ).
bIsClient	True if QUTS is a TCP client, false if QUTS is a TCP server.
Description	A human readable description of the connection.
Host	Host name or IP address. If bIsClient is true, this can be empty.
Port	Port on which to open the TCP connection.

## Returns

The instance of the protocol that was created, including the new ProtocolHandle and DeviceHandle for usage with QUTS APIs. See [ProtocolInfo](#).

### 4.3.2.11 removeTcpConnection()

Removes a TCP connection from QUTS. This is simply to clean up calls to [addTcpConnection\(\)](#).

## Prototype

```
Common.ErrorCode removeTcpConnection(1:i64 protocolHandle)
```

**Parameters**

protocolHandle	Protocol instance to remove
----------------	-----------------------------

**Returns**

None.

**4.3.2.12 startTcpServer()**

Starts a TCP server to listen for incoming connection requests.

**Prototype**

```
Common.ErrorCode startTcpServer(1:Common.ProtocolType protocolType,  
2:i32 port)
```

**Parameters**

protocolType	Type of protocol to use ProtocolType.
port	The port on which the server should poll for clients.

**Returns**

None.

**4.3.2.13 stopTcpServer()**

Stops the TCP server on the specified port.

**Prototype**

```
Common.ErrorCode stopTcpServer(1:i32 port)
```

**Parameters**

port	Port being used to service the server.
------	--

**Returns**

None.

**4.3.2.14 getDeviceBuildId ()**

Returns a structure containing build details for a device.

**Prototype**

```
Common.BuildInfo getDeviceBuildId(1:i64 deviceHandle)
```

### Parameters

deviceHandle	Handle of the device .
--------------	------------------------

### Returns

Structure containing details about the build on the device.

#### 4.3.2.15 getChipName ()

Returns the chip name of the device.

### Prototype

```
string getChipName(1:i64 deviceHandle,2:i64 protocolHandle)
```

### Parameters

deviceHandle	Handle of the device .
protocolHandle	Protocol handle of the diag protocol associated with the device. 0 can be passed if only one diag service is running on the device. If more than one diag protocol is present on the device and 0 is passed for protocolHandle, then the result will contain all the chip names of the diag protocols, each separated by "/".

### Returns

String containing the name of the chip.

#### 4.3.2.16 startLogging()

Initializes the session to begin saving log files at the current time.

### Prototype

```
Common.ErrorCode startLogging()
```

### Detailed description

Calling startLogging() is required to save logs (calling either saveLogFiles or saveLogFilesWithFileNames). Calling more than once does not do anything.

### Returns

None.

#### 4.3.2.17 saveLogFiles()

Saves the log files from all protocols that are used or were used by any service in the client session.

##### Prototype

```
list<string> saveLogFiles(1:string saveFolder)
```

##### Parameters

saveFolder	Location in which to save all log files.
------------	--

##### Detailed description

Saving the file saves everything from the last save point until the function is called. Therefore, saving the file multiple times causes multiple fragments, each containing the span between saves.

This will throw an exception if the client did not call startLogging.

##### Returns

A list of log files that were saved by this function.

#### 4.3.2.18 saveLogFilesWithFileNames()

Saves the log files from all protocols that are used or were used by any service in the client session.

##### Prototype

```
list<string> saveLogFilesWithFileNames(1:map<i64,string> logNamesByHandle)
```

##### Parameters

logNamesByHandle	Map of log file names per protocol handle.
------------------	--

##### Detailed description

This API works similar to the saveLogFiles API, but it will only save logs for the protocol handles specified in the map.

##### Returns

A list of log files that were saved by this function.

#### 4.3.2.19 resetLogFiles()

Starts new log files.

##### Prototype

```
Common.ErrorCode resetLogFiles()
```

##### Detailed description

Closes the current log file and starts a new one for each connected protocol. This will throw an exception if the client did not call startLogging.

##### Returns

None.

#### 4.3.2.20 logAnnotation()

Writes the annotation to the specified log file(s).

##### Prototype

```
Common.ErrorCode logAnnotation(1:string annotation,  
                                2:i64 messageId,  
                                3:i64 protocolHandle)
```

##### Parameters

annotation	Annotation text to log.
------------	-------------------------



messageId	Id to associate with the annotation. (useful for filtering)
protocolHandle	The protocol this annotation should be logged with, or -1 to log into all protocol logs.

**Returns**

None.

#### 4.3.2.21 openLogSession()

Creates a log session service to allow post processing of saved QUTS log sessions.

##### Prototype

```
string openLogSession(1:list<string> logFiles)
```

##### Parameters

logFiles	A list of log files to load into the session. Logs must be from the same original logging session or the API will fail.
----------	---

##### Detailed description

Calling openLogSession creates a new Thrift service of a log session service. The name of this service is returned, on which a log session service object must be used to communicate with the log session service's APIs. For usage, see [Postprocessing](#) log sessions.

##### Returns

The name of the log session service on which to create the Thrift service object.

#### 4.3.2.22 resetPhone()

Resets the device and returns true if the device is up before the reset timeout. QUTS will attempt to reset the device based on what protocols are available. It will first attempt to reset using any available diag protocols. If that is unsuccessful it will attempt to reset using ADB, Sahara, or Fastboot depending on which protocol is available at the time the API is called.

##### Prototype

```
Common.ErrorCode resetPhone(1:i64 deviceHandle, 2:i32 resetTimeoutMs)
```

##### Parameters

deviceHandle	Handle of the device
resetTimeoutMs	QUTS will wait for this duration (in milliseconds) for the device to come back up. If the device comes back up successfully by then, it will return TRUE; otherwise FALSE. Setting this value as 0 will make the API return immediately.

##### Returns

None.

#### 4.3.2.23 getThroughputStatistics()

Returns the byte and packet counts and the instantaneous throughput of the given protocol.

##### Prototype

```
Common.RxTxInfo getThroughputStatistics(1:i64 protocolHandle)
```

##### Parameters

protocolHandle	Handle of the protocol for which to get statistics.
----------------	---

##### Returns

RxTxInfo structure with throughput info (See [RxTxInfo](#)).

#### 4.3.2.24 getCurrentLogFileSize()

Returns the log file size of the given protocol in bytes.

##### Prototype

```
i64 getCurrentLogFileSize(1:i64 protocolHandle)
```

##### Parameters

protocolHandle	Handle of the protocol for which to get log file size.
----------------	--

##### Returns

Size of the current log file for the current client in bytes.

#### 4.3.2.25 getThroughput ()

Returns details about the rate, number of bytes, and number of packets received and sent.

##### Prototype

```
Common.RxTxInfo getThroughput(1:i64 protocolHandle)
```

##### Parameters

protocolHandle	Protocol whose details are required.
----------------	--------------------------------------

### Detailed description

For a protocol, the API would return the structure RxTxInfo filled out containing the details of number of packets received and sent, the number of bytes received and sent since connected, and the rate at which the packets were received in the last 0.5 sec interval.

### Returns

[RxTxInfo](#) structure.

#### 4.3.2.26 checkSpc()

Checks if the given SPC is correct or not on the device.

### Prototype

```
SpcStatus checkSpc(  
1:i64 deviceHandle,  
2:i64 diagProtocolHandle,  
3:string spc  
)
```

### Parameters

deviceHandle	Device whose details are required.
diagProtocolHandle	diagProtocol on which to check the SPC. Can pass in 0 if there is only a single diag handle present, and the lone diag protocol available will be selected as default.
Spc	SPC to test

### Detailed description

Helps to check if the entered SPC on the device is correct or not.

### Returns

[SpcStatus](#) struct with status set to true if the SPC is correct, else false.

#### 4.3.2.27 getOperatingMode()

Gets the operating mode.

### Prototype

```
OperatingModeInfo getOperatingMode(  
1:i64 deviceHandle,  
2:i64 diagProtocolHandle  
)
```

### Parameters

deviceHandle	Device whose details are required.
diagProtocolHandle	diagProtocol on which to check the SPC. Can pass in 0 if there is only a single diag handle present, and the lone diag protocol available will be selected as default.

### Detailed description

Gets the operating mode.

### Returns

[OperatingModeInfo](#)

### 4.3.2.28 setOperatingMode()

Changes the operating mode of the device diag handle to the ones defined in [OperatingMode](#).

### Prototype

```
Common.ErrorCode setOperatingMode(1:i64 deviceHandle,  
2:i64 diagProtocolHandle, 3:OperatingMode mode)
```

### Parameters

deviceHandle	Handle of the device
diagProtocolHandle	Diag protocol handle. If there is single diag protocol, can pass in 0 to select it as default.
Mode	The operating mode to set the device to. See <a href="#">OperatingMode</a> : The last three values in OperatingMode (OFFLINE, RESET and Net_TEST_GW) are not valid for setOperatingMode.

### Detailed description

Sets the operating mode. This is not supported on all devices; the device must have a diag protocol and must support command code 41. Some devices (for instance fusion) may not support this command, in which case an error will be returned.

### Returns

None

#### 4.3.2.29 setImei()

Sets the Imei on the device.

##### Prototype

```
Common.ErrorCode setImei(1:i64 deviceHandle, 2:i64 protocolHandle,  
3:binary imei, 4:i32 subscriptionId)
```

##### Parameters

deviceHandle	Handle of the device
diagProtocolHandle	Diag protocol handle. If there is single diag protocol, can pass in 0 to select it as default.
imei	The 9 bytes Imei value to set
subscriptionId	SIM to write to. Default: NO_SUBSCRIPTION_ID.

##### Returns

None

#### 4.3.2.30 getImei()

Gets Imei of the device.

##### Prototype

```
binary getImei(1:i64 deviceHandle, 2:i64 protocolHandle, 3:i32  
subscriptionId)
```

##### Parameters

deviceHandle	Handle of the device
diagProtocolHandle	Diag protocol handle. If there is single diag protocol, can pass in 0 to select it as default.
subscriptionId	SIM to write to. Default: NO_SUBSCRIPTION_ID.

##### Returns

Binary (byte array) representing Imei value.

#### 4.3.2.31 setMeid()

Sets the Meid on the device.

##### Prototype

```
Common.ErrorCode setMeid(1:i64 deviceHandle, 2:i64 protocolHandle,  
3:i64 meid, 4:i32 subscriptionId)
```

##### Parameters

deviceHandle	Handle of the device
diagProtocolHandle	Diag protocol handle. If there is single diag protocol, can pass in 0 to select it as default.
meid	The meid value to set
subscriptionId	SIM to write to. Default: NO_SUBSCRIPTION_ID.

##### Returns

None.

#### 4.3.2.32 getMeid()

Gets the Meid of the device.

##### Prototype

```
i64 getMeid(1:i64 deviceHandle, 2:i64 protocolHandle, 3:i32  
subscriptionId)
```

##### Parameters

deviceHandle	Handle of the device.
diagProtocolHandle	Diag protocol handle. If there is single diag protocol, can pass in 0 to select it as default.
subscriptionId	SIM to write to. Default: NO_SUBSCRIPTION_ID.

##### Returns

Meid of device.

#### 4.3.2.33 getActiveLogSession()

Gets the current active log session. Used for live processing.

##### Prototype

```
logSession getActiveLogSession()
```

##### Parameters

None

##### Returns

The logSession service handle associated with current active log session.

#### 4.3.2.34 getProtocolLockStatus ()

Gets the lock status of a protocol.

##### Prototype

```
Common.LockStatus getProtocolLockStatus(1:i64 protocolHandle)
```

##### Parameters

protocolHandle	Protocol handle.
----------------	------------------

##### Detailed description

Gets the detailed lock information of a protocol including lock state, locking client and reason.

##### Returns

[LockStatus](#)

#### 4.3.2.35 enableFunctionLog ()

Enable logging in certain functional areas.

##### Prototype

```
Common.ErrorCode enableFunctionLog(1:i64 deviceHandle,  
2:list<Common.FunctionArea> areas, 3:Common.LogOptions options)
```



## Parameters

deviceHandle	Handle of the device.
areas	List of functional areas to be logged.
options	<p>Logging options.</p> <p>Logging level, logging format and logging layout are not applicable. Each functional area has its own preset to maintain consistent file format for parsing purpose.</p> <p>If not specified,</p> <ul style="list-style-type: none"><li>■ Default log file size is 100,00KB</li><li>■ Default save log path is "C:\ProgramData\Qualcomm\QUTS\Logs".</li></ul>

## Detailed description

Once enabled, logging will remain active until QUTS exits or [disableFunctionLog](#) is called on the same device.

## Returns

None.

### 4.3.2.36 disableFunctionLog ()

Disable logging in certain functional areas.

## Prototype

```
Common.ErrorCode disableFunctionLog(1:i64 deviceHandle,  
2:list<Common.FunctionArea> areas)
```

## Parameters

deviceHandle	Handle of the device.
areas	List of functional areas to be logged.

## Returns

None.

### 4.3.2.37 enableProtocolLog ()

Enable logging for a protocol.

## Prototype

```
Common.ErrorCode enableProtocolLog(1:i64 protocolHandle,  
2:Common.LogOptions options)
```

## Parameters

deviceHandle	Handle of the device.
options	Logging options.  If not specified, <ul style="list-style-type: none"><li>■ Default logging format is LOG_CSV</li><li>■ Default logging level mask is LOG_Data only</li><li>■ Default layout is LOG_DATETIME, LOG_MESSAGE, LOG_LEVEL, LOG_LOGGER, LOG_CALLERFILE, LOG_CALLERMETHOD, LOG_CALLERLINE and LOG_TID</li><li>■ Default log file size is 100,00KB</li><li>■ Default save log path is "C:\ProgramData\Qualcomm\QUTS\Logs".</li></ul>

## Detailed description

Once enabled, logging will remain active until QUTS exits or [disableProtocolLog](#) is called on the same protocol.

## Returns

None.

### 4.3.2.38 disableProtocolLog ()

Disable logging for a protocol.

## Prototype

```
Common.ErrorCode disableProtocolLog (1:i64 protocolHandle)
```

## Parameters

deviceHandle	Handle of the device.
--------------	-----------------------

## Returns

None.

### 4.3.2.39 configureProtocol ()

Configure protocol (e.g. NMEA, RUMI diag device)

## Prototype

`Common.ErrorCode configureProtocol(1:Common.ProtocolConfiguration protocolConfiguration) throws (1:Common.AppException e)`

## Parameters

<b>protocolConfiguration</b>	Protocol configuration structure specifying parameters for the protocol configuration. See <a href="#">ProtocolConfiguration</a>
------------------------------	--

## Returns

None.

## 4.4 ClientCallback interface

This interface provides functions that QUTS can call to notify the client of changes or query the client for input. To receive this information, the client must implement this interface and create a Thrift multiplexed server with a single service called QUTS callback on port client port + 1.

### 4.4.1 Types

#### 4.4.1.1 MessageLevel

Provides the level of a message that returns through [onMessage\(\)](#).

```
enum MessageLevel
{
    INFO          = 0,
    WARNING       = 1,
    EXCEPTION     = 2
}
```

#### 4.4.1.2 YesNo

Provides the answers available for `getYesOrNo()`.

```
enum YesNo
{
    NO = 0,
    YES = 1
}
```

**Commented [KF1]:** This function was linked but the link was broken. The function does not seem to be listed in this doc at all. Should it be added? Or just remove this link?

#### 4.4.1.3 OkCancel

Provides the answers available for `getOkOrCancel()`.

```
enum OkCancel
{
    CANCEL = 0,
    OK = 1
}
```

**Commented [KF2]:** This function was linked but the link was broken. The function does not seem to be listed in this doc at all. Should it be added? Or just remove this link?

#### 4.4.1.4 ConnectionState

Enumerates the possible states of an opened connection. Used when signaling `onProtocolStateChange()`.

```
enum ConnectionState
{
    STATE_AVAILABLE = 0,
    STATE_DISCONNECTED = 1,
    STATE_UNRESPONSIVE = 2,
}
```

#### 4.4.1.5 RxTxInfo

Return type for the request to get the [RxTxInfo](#).

```
struct RxTxInfo
{
    1: required  i64  rxCount;
    2: required  i64  txCount;
    3: required  i64  rate;
    4: required  i64  byteRxCount;
    5: required  i64  byteTxCount;
}
```

##### Fields

rxCount	The number of packets received.
txCount	The number of packets sent.
rate	The rate at which the packets were sent and received.
byteRxCount	The bytes of data received.
byteTxCount	The bytes of data sent.

### 4.4.2 Functions

#### 4.4.2.1 onMessage()

Sends a message regarding the state of the QUTS server.

##### Prototype

```
oneway void onMessage(1:MessageLevel level,
                      2:string location,
                      3:string title,
                      4:string description)
```

##### Parameters

Level	<a href="#">MessageLevel</a> of the information provided.
Location	Place from which the message originated.
Title	Type of information contained.
Description	Actual message being conveyed.

##### Returns

None.

#### 4.4.2.2 onDeviceConnected()

Notifies the client that a device was connected.

##### Prototype

```
oneway void onDeviceConnected(1:i64 deviceHandle)
```

##### Parameters

deviceHandle	Device that was connected.
--------------	----------------------------

##### Returns

None.

#### 4.4.2.3 onDeviceDisconnected()

Notifies the client that a device was disconnected.

##### Prototype

```
oneway void onDeviceDisconnected(1:i64 deviceHandle)
```

##### Parameters

deviceHandle	Device that was disconnected.
--------------	-------------------------------

##### Returns

None.

#### 4.4.2.4 onDeviceModeChange()

Notifies the client about a mode change for a device.

##### Prototype

```
oneway void onDeviceModeChange(1:i64 deviceHandle,  
                                2:DeviceMode newMode)
```

##### Parameters

deviceHandle	Device that changed mode.
newMode	New mode of the device.

##### Returns

None.

#### 4.4.2.5 onProtocolAdded()

Notifies the client that a protocol was added to an existing device.

##### Prototype

```
oneway void onProtocolAdded(1:i64 deviceHandle,  
                             2:i64 protocolHandle)
```

##### Parameters

deviceHandle	Device on which the protocol was added.
protocolHandle	Protocol that was added.

##### Returns

None.

#### 4.4.2.6 onProtocolRemoved()

Notifies the client that a protocol was removed from an existing device.

##### Prototype

```
oneway void onProtocolRemoved(1:i64 deviceHandle,  
                               2:i64 protocolHandle)
```

##### Parameters

deviceHandle	Device on which the protocol was removed.
protocolHandle	Protocol that was removed.

##### Returns

None.

#### 4.4.2.7 onProtocolStateChange()

Notifies the client about a state change for an opened protocol.

##### Prototype

```
oneway void onProtocolStateChange(1:i64 protocolHandle,  
                                   2:ProtocolState newState)
```

##### Parameters

protocolHandle	Protocol that changed state.
newState	New state of the protocol.

**Returns**

None.

**4.4.2.8 onProtocolFlowControlStatusChange()**

Notifies the client about a flow control status change for an opened protocol.

**Prototype**

```
oneway void onProtocolFlowControlStatusChange(1:i64 protocolHandle,  
                                              2:Common.Direction dir,  
                                              3:Common.FlowControlStatus newStatus)
```

**Parameters**

protocolHandle	Protocol that has a flow control status change.
dir	Flow control direction
newStatus	New flow control status of the protocol for the given direction.

**Returns**

None.

**4.4.2.9 onProtocolLockStatusChange ()**

Notifies the client about a lock status change for an opened protocol.

**Prototype**

```
oneway void onProtocolLockStatusChange(1:i64 protocolHandle,  
                                       2:Common.LockStatus newStatus)
```

**Parameters**

protocolHandle	Protocol that has a lock status change.
newStatus	New lock status of the protocol.

**Returns**

None.

**4.4.2.10 onAsyncResponse()**

Notifies the client about a response being received for an asynchronous request.

**Prototype**

```
oneway void onAsyncResponse(1:i64 protocolHandle,  
                           2:i64 transactionId)
```



**Parameters**

protocolHandle	Protocol that received the response.
transactionId	Transaction ID of the request.

**Returns**

None.

**4.4.2.11 onDataQueueUpdated()**

Signals when a data queue has new data to be retrieved. This function is used in conjunction with [createDataQueue\(\)](#).

**Prototype**

```
oneway void onDataQueueUpdated(1:string queueName, 2:i32 queueSize)
```

**Parameters**

queueName	Name of the queue that has new items.
queueSize	The number of items currently in the queue.

**Detailed description**

This function is called once every 50 milliseconds at most, even if multiple items were added during that duration.

Note: queueSize is valid at the time the calculation of data queue is initiated but not necessarily by the time the user receives the event. The callback only works as an indicator that the user should check for new data.

**Returns**

None.

**4.4.2.12 onServiceAvailable()**

Signals that a service has become available on a device.

**Prototype**

```
oneway void onServiceAvailable(1:string serviceName,  
                                2:i64 deviceHandle)
```

**Parameters**

serviceName	Service that became available.
deviceHandle	Device on which the service is available.

**Returns**

None.

**4.4.2.13 onServiceEnded()**

Signals that a service is no longer available on a device.

**Prototype**

```
oneway void onServiceEnded(1:string serviceName,  
                           2:i64 deviceHandle)
```

**Parameters**

serviceName	Service that ended.
deviceHandle	Device on which the service ended.

**Returns**

None.

**4.4.2.14 onServiceEvent()**

Callback for an event that occurred in one of the Thrift services. The actual payload of the parameters is defined per service.

**Prototype**

```
oneway void onServiceEvent(1:string serviceName,  
                           2:i64 eventId,  
                           3:string eventDescription)
```

**Parameters**

serviceName	Service in which the event occurred.
eventId	Service-defined ID of the event.
eventDescription	Optional description of the event.

**4.4.2.15 onQShrinkStateUpdated()**

Callback indicating that the state of loading a QShrink4 hash file has changed in QUTS for the given protocol.

**Prototype**

```
oneway void onQShrinkStateUpdated(1:i64 protocolHandle,  
2:Common.QShrink4DownloadState newState)
```

### Parameters

protocolHandle	Protocol instance that has an updated state regarding QShrink4 loading
newState	The current load state for QShrink4 on that protocol (See <a href="#">QShrink4DownloadState</a> ).

#### 4.4.2.16 onDataViewUpdated()

Signals when a data view has new data to be retrieved. This function is used in conjunction with [createDataView\(\)](#).

### Prototype

```
oneway void onDataViewUpdated(1:string viewName, 2:i32 viewSize, 3:bool finished),
```

### Parameters

viewName	Name of the data view that has new items.
viewSize	The number of items currently in the data view.
finished	True when the data view is done loading file and all items are available.

### Detailed description

This function is called once every 50 milliseconds at most, even if multiple items were added during that duration.

### Returns

None.

#### 4.4.2.17 onMissingQShrinkHashFile()

Signals when a QShrink4 file with particular GUID is not found or not loaded for decoding in the live session. Users can listen to this call back and then load a qshrink file, see [loadQShrinkFile \(\)](#).

### Prototype

```
oneway void onMissingQShrinkHashFile(1:i64 protocolHandle, 2:string missingFileGuid)
```

### Parameters

protocolHandle	Diag protocol which was missing the QShrink hash file
missingFileGuid	Missing qshrink4 file with particular guid

### Returns

None.

#### 4.4.2.18 onLogSessionMissingQShrinkHashFile()

Signals when a QShrink4 file with particular GUID is not found or not loaded for decoding in a post processing log session. Users can listen to this call back and then load a QShrink file. See loadQShrinkFile() on Log Session.

#### Prototype

```
oneway void onLogSessionMissingQShrinkHashFile(1:string
logSessionInstance, 2:i64 protocolHandle, 3:string missingFileGuid)
```

#### Parameters

logSessionInstance	Which log session is missing the QShrink file. This is the name generated when opening a log session in Device Manager openLogSession() <a href="#">4.3.2.21</a>
protocolHandle	Diag protocol which was missing the QShrink hash file
missingFileGuid	Missing qshrink4 file with particular guid

#### Returns

None.

## 4.5 LogSession interface

A LogSession is designed to allow postprocessing log files that were generated by a previous instance of QUTS. The LogSession recombines the logs to be processed in the correct order in which items were received, but all logs must have originated from the same run of QUTS. To create a LogSession, see [openLogSession\(\)](#).

### 4.5.1 Types

#### 4.5.1.1 IndexType

Specifies whether an index or range of indexes should refer to the protocol, session, or data view index.

```
enum IndexType
{
    PROTOCOL_INDEX = 0,
    SESSION_INDEX = 1,
    DATA_VIEW_INDEX = 2
}
```

#### Index types

PROTOCOL_INDEX	Indexes for a specific instance of an index
SESSION_INDEX	Global index across all instances of all protocols that are connected

DATA_VIEW_INDEX	Index relative to the specified filtered subset of packets
-----------------	--

#### 4.5.1.2 PacketRange

Specifies a range of packets. Only the endIndex or count field is used at a time.

```
struct PacketRange
{
    1: optional IndexType indexType;
    2: optional i64 beginIndex;
    3: optional i64 endIndex;
    4: optional i64 count;
}
```

##### Fields

indexType	If different indexes could be used, this specifies which one to create a range on (see <a href="#">IndexType</a> ).
beginIndex	Specifies the first item in the range to return.
endIndex	Specifies the last item in the range.
Count	Gives the number of items to return.

#### 4.5.1.3 DataPacketFilter

Aggregates all possible packet filter items.

```
struct DataPacketFilter
{
    1: list<i64> protocolHandleList;
    2: optional map<i64, PacketRange> protocolRange;
    3: optional Common.DiagPacketFilter diagFilter;
    4: optional Common.QmiPacketFilter qmiFilter;
    5: optional Common.AdbPacketFilter adbFilter;
    6: optional Common.SaharaPacketFilter saharaFilter;
    7: optional Common.FastbootPacketFilter fastbootFilter;
    8: optional Common.NmeaPacketFilter nmeaFilter;
    9: optional Common.AnnotationPacketFilter annotationsFilter;
    10: optional bool registerMaskToDevice
}
```

##### Fields

protocolHandleList	List of protocols to include in this filter.
protocolRange	The range of packets to retrieve for each protocol. If empty all packets will be retrieved.
diagFilter	Diag-specific filter (see <a href="#">DiagPacketFilter</a> ).
qmiFilter	QMI-specific filter (see <a href="#">QmiPacketFilter</a> ).

<code>adbFilter</code>	ADB-specific filter (see <a href="#">AdbPacketFilter</a> ).
<code>saharaFilter</code>	Sahara-specific filter (see <a href="#">SaharaPacketFilter</a> ).
<code>fastBootFilter</code>	Fastboot-specific filter (see <a href="#">FastbootPacketFilter</a> ).
<code>nmeaFilter</code>	NMEA-specific filter (see <a href="#">NmeaPacketFilter</a> ).
<code>annotationsFilter</code>	Annotation-specific filter (see <a href="#">AnnotationPacketFilter</a> ).
<code>registerMaskToDevice</code>	Specifies whether the masks in the filters need to be applied to device or not. Default is false.

#### 4.5.1.4 DataPacket

Aggregates all possible data packets.

```
struct DataPacket
{
    1: i64 protocolHandle;
    2: optional Common.DiagPacket diagPacket;
    3: optional Common.QmiPacket qmiPacket;
    4: optional Common.AdbPacket adbPacket;
    5: optional Common.SaharaPacket saharaPacket;
    6: optional Common.FastbootPacket fastbootPacket;
    7: optional Common.AdplPacket adplPacket;
    8: optional Common.NmeaPacket nmeaPacket;
    9: optional Common.AnnotationPacket annotationPacket;
}
```

#### Fields

protocolHandle	Which protocol the packet came from
diagPacket	Populated if protocolHandle is a diag protocol. See <a href="#">DiagPacket</a> .
qmiPacket	Populated if protocolHandle is a QMI protocol. See <a href="#">QmiPacket</a> .
adbPacket	Populated if protocolHandle is a ADB protocol. See <a href="#">AdbPacket</a> .
saharaPacket	Populated if protocolHandle is a Sahara protocol. See <a href="#">SaharaPacket</a> .
fastbootPacket	Populated if protocolHandle is a FastBoot protocol. See <a href="#">FastbootPacket</a> .
adplPacket	Populated if protocolHandle is a ADPL protocol. See <a href="#">AdplPacket</a> .
nmeaPacket	Populated if protocolHandle is a Nmea protocol. See <a href="#">NmeaPacket</a> .
annotationPacket	Populated if Annotation filters are enabled. See <a href="#">AnnotationPacket</a> .

#### 4.5.1.5 PacketReturnConfig

Aggregates all possible return config structures.

```
struct PacketReturnConfig
{
    1: optional Common.DiagReturnConfig diagConfig;
    2: optional Common.QmiReturnConfig qmiConfig;
    3: optional Common.AdbReturnConfig adbConfig;
    4: optional Common.SaharaReturnConfig saharaConfig;
}
```

## Fields

diagConfig	Configuration for any Diag protocols. See <a href="#">DiagReturnConfig</a> .
qmiConfig	Configuration for any QMI protocols. See <a href="#">QmiReturnConfig</a> .
adbConfig	Configuration for any ADB protocols. See <a href="#">AdbReturnConfig</a> .
saharaConfig	Configuration for any Sahara protocols. See <a href="#">SaharaReturnConfig</a> .
nmeaConfig	Configuration for any Nmea protocols. See <a href="#">NmeaReturnConfig</a> .

### 4.5.1.6 ViewConfigurationUpdate

Type of update operation on filters of existing log session.

```
enum ViewConfigurationUpdate
{
    ADD_NEW_CONFIG_ITEMS = 0,
    REPLACE_CURRENT_CONFIG_ITEMS = 1,
    REMOVE_SELECTED_CONFIG_ITEMS = 2
}
```

### 4.5.1.7 DataViewServiceEvent

Event type to notify data view updates..

```
enum DataViewServiceEvent
{
    INITIALIZE = 0,
    SAVE_ITEMS_BY_VIEW_INDEX = 1
}
```

### 4.5.1.8 DataViewInfo

Contains the information for creating a data type. Used in an array in createDataViewSet to create multiple views at a single time for better performance.

```
struct DataViewInfo
{
    1:string viewName;
    2:DataPacketFilter filter;
    3:PacketReturnConfig returnConfig;
}
```



## Fields

viewName	The name of a view. Must be unique within this LogSession.
filter	Packet filter. See 4.5.1.3 .
returnConfig	Default return configuration for the view. See 4.5.1.5 .

## 4.5.2 Functions

### 4.5.2.1 getLastError()

Queries for the error from the last function call.

#### Prototype

```
Common.ErrorType getLastError()
```

#### Returns

Error code and description.

### 4.5.2.2 destroyLogSession()

Removes the current log session from QUTS.

#### Prototype

```
Common.ErrorCode destroyService()
```

#### Returns

None.

### 4.5.2.1 getDeviceList()

Gets the information for all devices that were connected in the log session.

#### Prototype

```
list<Common.DeviceInfo> getDeviceList()
```

#### Returns

List of devices (see [DeviceInfo](#)).

Upon a failure, the list is empty (for the error code, call [getLastError\(\)](#)).

### 4.5.2.2 getProtocolList()

Gets a list of protocols that were logged on a specified device.

## Prototype

```
list<Common.ProtocolInfo> getProtocolList(1:i64 deviceHandle)
```

## Parameters

deviceHandle	Device to get protocols for.
--------------	------------------------------

## Returns

List of available protocols (see [ProtocolInfo](#)).

Upon a failure, the list is empty (for the error code, call [getLastError\(\)](#)).

### 4.5.2.3 getDataPacketCount()

Gets the number of packets logged by the specified protocol in the session.

## Prototype

```
i64 getDataPacketCount(1:i64 protocolHandle)
```

## Returns

The number of packets in the given protocol.

Upon failure, returns 0 (for the error code, call [getLastError\(\)](#)).

### 4.5.2.4 createDataView()

Creates a view over a subset of packets based on the filter definition. For live log sessions, the registerMaskToDevice parameter in the DataPacketFilter can be used to specify if the filters should also be used to set mask on device. Default is False.

## Prototype

```
Common.ErrorCode createDataView(1:string viewName,  
                                2:DataPacketFilter filter,  
                                3:PacketReturnConfig returnConfig)
```

## Parameters

viewName	A unique view name within this log session
filter	What packets to include in this view
returnConfig	What information to return when returning data items in <code>getDataViewItems</code> / <code>getDataViewItemCount()</code> .

### 4.5.2.5 createDefaultDataView()

Creates a view over all packets from a given set of protocols. This processing can become very expensive; during high data rate this view could cause processing delays

or increased memory consumption. It is recommended to avoid using this API and instead use `createDataView()` with a filtered subset specifying exactly what is desired.

### Prototype

```
Common.ErrorCode createDefaultDataView(1:string viewName,  
    2:list<i64> protocolHandleList,  
    3:PacketReturnConfig returnConfig)
```

### Parameters

viewName	A unique view name within this log session
protocolHandleList	What protocols to gather packets from
returnConfig	What information to return when returning data items in <code>getDataViewItemCount()</code> .

#### 4.5.2.6 removeDataView()

Removes a view created in [createDataView\(\)](#).

### Prototype

```
Common.ErrorCode removeDataView(1:string viewName)
```

### Parameters

viewName	The view to remove from QUTS.
----------	-------------------------------

#### 4.5.2.7 getDataViewItemCount()

Gets a number of items currently in the given view.

### Prototype

```
i32 getDataViewItemCount(1:string viewName)
```

### Parameters

viewName	The name of the view.
----------	-----------------------

### Returns

The number of items in the given view.

#### 4.5.2.8 getDataViewItems()

Gets a range of items from the given view. The return information is specified in the queue creation to avoid redundant passing/parsing of the information during each call to `getDataViewItems()`.

## Prototype

```
list<DataPacket> getDataViewItems(1:string viewName, 2:PacketRange
packets)
```

## Parameters

viewName	The name of the view to retrieve items from.
packets	The range of packets to retrieve from the view (see <a href="#">PacketRange</a> ).

## Returns

A list of DataPackets from the view (see [DataPacket](#)).

Upon failure, returns an empty list (for the error code, call [getLastError\(\)](#)).

### 4.5.2.9 getDataViewItemsForConfiguration()

This API is the same as [getDataViewItems](#), except it allows overriding the return config for just this API call.

## Prototype

```
list<DataPacket> getDataViewItems(1:string viewName, 2:PacketRange
packets, 3:PacketReturnConfig returnConfig)
```

## Parameters

viewName	The name of the view to retrieve items from.
packets	The range of packets to retrieve from the view (see <a href="#">PacketRange</a> ).
returnConfig	What information to return when returning data items.

## Returns

A list of DataPackets from the view (see [DataPacket](#)).

Upon failure, returns an empty list (for the error code, call [getLastError\(\)](#)).

### 4.5.2.10 createDefaultDataView()

Creates a default view without any filtering of the packets.

## Prototype

```
Common.ErrorCode createDefaultDataView(1:string viewName,
2: list<i64> protocolHandleList,
3:PacketReturnConfig returnConfig)
```

## Parameters

viewName	A unique view name within this log session.
----------	---

protocolHandleList	List of protocol handles to open in the log session.
returnConfig	What information to return when returning data items in <code>getDataViewItemCount()</code> .

#### 4.5.2.11 saveDataViewItemsByIndex()

Save data view items by indexes to a new HDFs at destination folder. API to copy the subset of items from data view to new HDFs.

##### Prototype

```
Common.ErrorCode saveDataViewItemsByIndex(1:string viewName,  
                                           2:string destinationFolder,  
                                           3:list<i64> dataViewIndexes)
```

##### Parameters

viewName	The view name.
destinationFolder	Target folder name to save the data view items. New HDF files will be created in this folder with selected items.
dataViewIndexes	List of data view item indexes to be copied to new folder in new HDF files.

#### 4.5.2.12 saveDataViewItemsByIndexWithFileNames()

This API works similar to the `saveDataViewItemsByIndex` API. This API will only save files for the protocol handles specified in the map.

##### Prototype

```
Common.ErrorCode saveDataViewItemsByIndexWithFileNames(  
    1:string viewName,  
    2:list<i64> dataViewIndexes,  
    3:map<i64,string> logNamesByHandle)
```

##### Parameters

viewName	The view name.
dataViewIndexes	List of data view item indexes to be copied to new folder in new HDF files.
logNamesByHandle	A map of log filenames by protocol handle.

#### 4.5.2.13 saveLogFiles()

Saves the converted log files from this log session.

##### Prototype

```
list<string> saveLogFiles(1:string saveFolder)
```

## Parameters

saveFolder	Location in which to save all log files.
------------	--

## Detailed description

This will save the entire session into hdf logs. This API is only meant for post-processing and will throw an exception if this is a live session. For live sessions, call DeviceManager.saveLogFiles().

## Returns

A list of log files that were saved by this function.

### 4.5.2.14 getDurationByProtocol()

## Prototype

```
string getDurationByProtocol(1:i64 protocolHandle)
```

## Parameters

protocolHandle	Handle of the protocol for which to get log duration
----------------	--

## Returns

The duration of the log session for a specific protocol

### 4.5.2.15 updatePacketFilters()

Update filters in current log session. Note: The registerMaskToDevice field in the DataPacketFilter filter doesn't take effect in this case and is ignored. If registerMaskToDevice needs to be set, it needs to be specified in createDataView or CreateDataViewSet only and the value will remain in effect for the duration of the dataView.

## Prototype

```
Common.ErrorCode updatePacketFilters(1:string viewName,  
                                     2:DataPacketFilter filter,  
                                     3:ViewConfigurationUpdate updateType)
```

## Parameters

viewName	A unique view name within this log session.
filter	New DataPacketFilter configuration (see <a href="#">DataPacketFilter</a> )
updateType	Type of update on the log session (see <a href="#">ViewConfigurationUpdate</a> )

#### 4.5.2.16 updatePacketReturnConfig()

Replace exiting packet return config with a new return configuration.

##### Prototype

```
Common.ErrorCode updatePacketReturnConfig(1:string viewName,  
                                           2:PacketReturnConfig returnConfig)
```

##### Parameters

viewName	A unique view name within this log session.
returnConfig	New return configuration (see <a href="#">PacketReturnConfig</a> )

#### 4.5.2.17 getAvailablePacketIds()

Get the packet IDs available/received on a protocol.

##### Prototype

```
DataPacketFilter getAvailablePacketIds(1:i64 protocolHandle)
```

##### Parameters

protocolHandle	Handle of the protocol for which to get log duration
----------------	--

##### Returns

The [DataPacketFilter](#) with packet information available on the protocol.

#### 4.5.2.18 loadQShrinkFile ()

Loads a QShrink file from given path.

##### Prototype

```
void loadQShrinkFile(1:i64 protocolHandle, 2:string path)
```

##### Detailed description

The QShrink file specified in the path is loaded and the symbols from that Qshrink file are used.

##### Returns

None.

#### 4.5.2.19 createDataSet()

Creates a set of data views to run in a batch to achieve better performance. This is the same as calling createDataSet (4.5.2.4 ) multiple times but allows a single disk access pointer to reduce thrashing and improve performance.

##### Prototype

```
Common.ErrorCode createDataSet(1:list<DataSetInfo> dataSetDefinitions)
```

##### Parameters

dataSetDefinitions	Any array of data view definitions. See 4.5.1.8 .
--------------------	---

##### Returns

None

#### 4.5.2.20 setWcdmaProtocolRevision()

Set WCDMA protocol revision value from the values in the enum [WcdmaProtocolRevision](#).

##### Prototype

```
Common.ErrorCode setWcdmaProtocolRevision(  
    1:i64 protocolHandle,  
    2:WcdmaProtocolRevision revision)
```

##### Detailed description

This function allows the user to set the WCDMA protocol revision that is used when parsing WCDMA OTA log items.

##### Returns

None.

#### 4.5.2.21 setCdmaProtocolRevision()

Set CDMA protocol revision value from the values in the enum [CdmaProtocolRevision](#).

##### Prototype

```
Common.ErrorCode setCdmaProtocolRevision(  
    1:i64 protocolHandle,  
    2:CdmaProtocolRevision revision)
```



### Detailed description

This function allows the user to set the CDMA protocol revision that is used when parsing CDMA OTA log items.

### Returns

None.

#### 4.5.2.22 setPilotInc()

Sets the value of Pilot inc. The value is within the range (0:5).

### Prototype

```
Common.ErrorCode setPilotInc(1:i64 protocolHandle, 2:i16 pilotInc)
```

### Returns

None.

## 4.6 UtilityService interface

A UtilityService handle can be obtained by calling GetUtilityService() on a QutsClient object in Perl/Python/C# or Java. The following functions are then available on the UtilityService handle obtained.

### 4.6.1 Types

#### 4.6.1.1 QmiUnpackReturn

Return type for QmiUnpack API containing the unpacked XML string containing the TLV and message name to which the TLV corresponds to.

```
struct QmiUnpackReturn
{
    1: optional string msgName;
    2: optional string tlvXml;
}
```

### Fields

msgName	The request/response/indication message name corresponding to the input bytes.
tlvXml	The unpacked TLVs in form of XML string

## 4.6.2 Functions

### 4.6.2.1 qmiPack()

Packs/converts a QMI message from XML format to bytes.

#### Prototype

```
binary qmiPack(  
    1:string serviceIdOrName,  
    2:string messageIdOrName,  
    3:string xmlRequest)
```

#### Parameters

serviceIdOrName	QMI service name or ID (for example "NAS" or "3")
messageIdOrName	QMI message name or ID (for example, QMI_NAS_GET_SIGNAL_STRENGTH_REQ")
xmlRequest	QMI message with inputs in XML format. For example: '<opt> <request_mask><request_mask>1</request_mask></request_mask> </opt>'

#### Returns

Binary output representing packed bytes for the given QMI message.

### 4.6.2.2 qmiUnPack()

Unpacks/converts a QMI message from bytes to XML format.

#### Prototype

```
Common.QmiUnpackReturn qmiUnPack(  
    1:string serviceIdOrName,  
    2:binary input  
)
```

#### Parameters

serviceIdOrName	QMI service name or ID (for example, "NAS" or "3")
input	Binary input representing packed bytes for a QMI message within the serviceId.

#### Returns

String representing QMI message in XML format.

#### 4.6.2.1 nvGetAllItems ()

Returns all NVs with ID, Name, Description and Category.

##### Prototype

```
list<Common.NvItem> nvGetAllItems ()
```

##### Parameters

None

##### Returns

List of [NvItems](#).

#### 4.6.2.2 nvGetItemDefintion ()

Returns the definition of the NV item from the ID or name.

##### Prototype

```
string nvGetItemDefintion(1:string nvItemNameOrId)
```

##### Parameters

nvItemNameOrId	NV item name or ID. Either the NV ID or legacy name.
----------------	--

##### Returns

JSON containing the definition of the NvItem.

#### 4.6.2.3 convertToHdf ()

Converts a list of qmdl2/bin files from the same log session and converts to one or more HDF log files.

##### Prototype

```
List<string> convertToHdf(1:list<string> logFiles, 2:string saveFolder)
```

##### Parameters

logFiles	List of files to convert to hdf.
saveFolder	Folder to save the converted log file(s).

##### Returns

The list of converted hdf log files.



## 5 QUTS services interfaces

---

All device functions are made available on the DeviceManager interface. If functions return a failure indicator, call [getLastError\(\)](#) to determine what caused the failure.

### 5.1 Device config service

The device config service provides functions for interacting with NV items, encrypted file system (EFS), and MCFG binaries (MBN) settings of the device. The service also provides support for backing up and restoring these items. This service attempts to lock the diag protocol so that it cannot be shared by other clients. If another client is already using the diag protocol with sharing enabled, then it shares with the existing client. This may cause inconsistent states and behaviors on the device. A warning is sent if this is the case. The QMI protocol is shared.

#### 5.1.1 Constants

##### 5.1.1.1 PDC\_CONFIGURATION\_VALUE\_INVALID

Returned as an error indication from [pdcGetConfiguration\(\)](#).

```
const i32 PDC_CONFIGURATION_VALUE_INVALID = 0xFFFFFFFF
```

##### 5.1.1.2 NO\_SUBSCRIPTION\_ID

Used in NV functions to indicate use of the default SIM card for operations.

```
const i32 NO_SUBSCRIPTION_ID = 0xFFFFFFFF
```

#### 5.1.2 Types

##### 5.1.2.1 PdcMbnType

Used in PDC functions to specify the type of configuration memory store on which to operate.

```
enum PdcMbnType
{
    PDC_CONFIG_TYPE_MODEM_PLATFORM = 0,
    PDC_CONFIG_TYPE_MODEM_SW = 1
}
```

#### 5.1.2.2 PdcConfigurationType

Specifies the configuration item to operate on in [pdcGetConfiguration\(\)](#) and [pdcSetConfiguration\(\)](#).

```
enum PdcConfigurationType
{
    SELECTION_MODE = 0,
    CARRIER       = 1,
    FLEX_MAPPING   = 2,
    REFRESH_MODE   = 3,
    MULTISIM       = 4,
    OS             = 5,
    MARKET         = 6,
    DEPLOYMENT     = 7,
    IMS_FEATURE    = 8,
    VARIANT        = 9,
    COUNTRY        = 10
}
```

#### 5.1.2.3 PdcMbnInfo

Returns information about a specified MBN.

```
struct PdcMbnInfo {
    1: Common.ErrorCode errorCode;
    2: optional i32 type;
    3: optional i32 size;
    4: optional i32 version;
    5: optional i32 baseVersion;
    6: optional i32 storageType;
    7: optional binary configId;
    8: optional string description;
    9: optional string path;
}
```

#### 5.1.2.4 PdcMbnResult

Returns validation information from [pdcValidateMbn\(\)](#).

```
struct PdcMbnResult {
    1: Common.ErrorCode errorCode;
    2: optional i32 version,
    3: optional string frame
}
```

### 5.1.2.5 NvReturnFlags

Flags for requesting the types of data to return when calling functions that return an [NvData](#) value.

```
enum NvReturnFlags
{
    BINARY_PAYLOAD = 0x00000001,
    PARSED_TEXT = 0x00000002,
    PARSED_JSON = 0x00000003,
    VALUE_LIST = 0x00000004
}
```

### 5.1.2.6 NvReturns

Specifies what information to return for NV read calls.

```
struct NvReturns
{
    1: NvReturnFlags flags;
    2: optional list<string> fieldQueries;
}
```

#### Fields

flags	What fields to return in resulting <a href="#">NvData</a> .
queries	Specifies queries to make for field values in the log packet. For information on queries, see <a href="#">Field</a> queries.

### 5.1.2.7 NvData

Return types for functions that request an [NvData](#) value.

This structure contains optional fields that can be filled using the [NvReturnFlags](#) type. All fields other than `errorCode` are available only if [ErrorCode](#) is `DEVICE_NO_ERROR (0)`. Other fields are optionally available depending on the values specified in [NvReturnFlags](#).

```
struct NvData
{
    1: Common.ErrorCode errorCode;
    2: optional binary payload;
    3: optional string parsedText;
    4: optional string parsedJson;
    5: optional string valueList;
    6: optional string queryResultJson;
}
```

## Fields

errorCode	Code if an error occurred. All other fields available only when <code>DEVICE_NO_ERROR (0)</code> is set in <a href="#">ErrorCode</a> .
payload	Raw binary payload of the response starting at the command code. Available when <code>BINARY_PAYLOAD</code> is set in <a href="#">NvReturnFlags.sendRawRequest()</a>
parsedText	Parsed text of the payload. Available when <code>PARSED_TEXT</code> is set in <a href="#">NvReturnFlags..</a>
parsedJson	Payload parsed into JSON format Available when <code>PARSED_JSON</code> is set in <a href="#">NvReturnFlags</a> .
valueList	Payload parsed into comma separated list format. Available when <code>VALUE_LIST</code> is set in <a href="#">NvReturnFlags</a> .
queryResultJson	Returns the results of any queries specified in <a href="#">NvReturnFlags</a> in a single JSON formatted string.

### 5.1.2.8 FileSystem

For all EFS functions, the `FileSystem` dictates whether the command should be executed on the primary file system or the alternate file system.

```
enum FileSystem
{
    FS_PRIMARY = 0,
    FS_ALTERNATE = 1
}
```

### 5.1.2.9 EfsType

Specifies the type of EFS entry being referred to in [EfsItem](#).

```
enum EfsType
{
    DIRECTORY = 0,
    FILE = 1
}
```

### 5.1.2.10 EfsItem

Contains directory entry information for items returned in [efsGetDirectoryContents\(\)](#).

```
struct EfsItem
{
    1: EfsType type;
    2: string name;
}
```



### 5.1.2.11 EfsFileAttributes

Return type for `efsGetFileAttributes()`

This structure contains optional fields that correspond to file attributes.

```
struct EfsFileAttributes
{
    1: string modifiedTime;
    2: string createTime;
    3: string attributes;
    4: i32    mode;
    5: i32    nLinks;
    6: i64    fileSize;
}
```

The attributes sub field in the `EfsFileAttributes` structure has a three letter string format that carries information about file attributes, buffering options, and cleanup options as follows:

- File attributes – First letter in the attributes string. It provides information about the attributes of the file. Possible values are:
  - U – Unrestricted
  - P – Permanent; cannot be removed or truncated
  - R – Read only
  - S – System permanent; file will remain after a system reformat
  - \* – Remote file; resides outside file system address space
  - \_ – Unknown, could not determine/received invalid response from device
- Buffering option – Second letter in the attributes string. It indicates whether to allow file buffering for the file. Possible values are:
  - P – Prohibit file buffering. Physical writes must be completed on the device before write operations are acknowledged
  - A – Allow file buffering. The EFS may use RAM to buffer written data to an open file. This increases the slight possibility of data loss in the event of an abnormal power down of the DMSS
  - \_ – Unknown. Could not determine/received invalid response from device
- Cleanup option – Third letter in the attributes string. Cleanup option in case of failure. Valid values are:
  - C – Close file as-is
  - D – Delete file and remove from directory
  - T – Truncate file from specified position
  - R – Revert to last check-pointed version
  - \_ – Unknown. Could not determine/received invalid response from device.

#### 5.1.2.12 DeviceConfigConnectionOptions

Supports special options for connections. This can be used if Diag is needed to be started with some special connection options. Note that DiagConnectionOptions has few different options that can be used when DiagService is created, but for DeviceConfigService, only the enableHdlcOnly option is supported among the options available in DiagConnectionOptions. Even if other options are passed in from DeviceConfigService, they will be ignored. If HDLC only mode is needed, enableHdlcOnly needs to be set by the first client creating diag service on Quts for that Quts instance. If not set firsttime, then Quts will switch to nonHdlc by default and it will remain in effect until the last diag client is disconnected. This enableHdlcOnly feature is available in both deviceconfigService and diagService initializeServiceWithOptions, so whichever service is first created, it needs to set this to true if HDLC only mode is desired.

```
struct DeviceConfigConnectionOptions
{
    1: optional DiagService.DiagConnectionOptions diagConnectionOptions;
    2: optional Common.QmiConnectionOptions qmiConnectionOptions;
}
```

#### 5.1.2.13 DeviceConfigOptions

Supports special options for DeviceConfigService if needed to be set after the connection. Currently only the option to use Qmi over Diag for Pdc Mbn is supported.

```
struct DeviceConfigOptions
{
    1: optional bool useDiagForPdc;
}
```

Quts will use Qmi for Pdc APIs (APIs starting with pdc in DeviceConfigService). And if Qmi is not available then Quts will use Diag to send the same PDC Qmi commands. Use this option (set to True) to use Diag only and not Qmi.

### 5.1.3 Functions

#### 5.1.3.1 getLastError()

Queries for the error from the last function call.

##### Prototype

```
Common.ErrorType getLastError()
```

##### Returns

Error code and description.

### 5.1.3.2 getDevice()

Gets the device handle on which this service was created.

#### Prototype

```
i64 getDevice()
```

#### Returns

Device handle.

### 5.1.3.3 initializeService()

Opens DIAG and QMI connections and starts a service.

If there is more than one diag protocol, this function will fail. For such devices, use [initializeServiceByProtocol\(\)](#) instead. If more than one QMI protocol is available on the device, QUTS tries to find a default QMI protocol handle by finding the one which does not have "MUX" in the description name. For example, for a given device which enumerates QMI as described below, QUTS would select "Qualcomm HS-USB WWAN Adapter 9035" for getting the QMI handle.

```
Qualcomm HS-USB WWAN Adapter 9035
Qualcomm HS-USB WWAN Adapter 9035-81 (MUX2)
Qualcomm HS-USB WWAN Adapter 9035-82 (MUX3)
Qualcomm HS-USB WWAN Adapter 9035-83 (MUX4)
Qualcomm HS-USB WWAN Adapter 9035-84 (MUX5)
Qualcomm HS-USB WWAN Adapter 9035-85 (MUX6)
Qualcomm HS-USB WWAN Adapter 9035-86 (MUX7)
Qualcomm HS-USB WWAN Adapter 9035-80 (MUX1)
```

Conversely, if more than one QMI protocol is present that has "MUX" in the description, QUTS would select any one of them for acquiring the QMI protocol handle.

#### Prototype

```
Common.ErrorCode initializeService()
```

#### Returns

None.

### 5.1.3.4 initializeServiceByProtocol()

Opens diag and QMI connections and starts a service. In the case that a device has multiple diag or QMI connections (such as fusion devices), this API must be used instead of `initializeService`.

Call [getProtocolList\(\)](#) on the device that is to be connected, determine the appropriate instances of diag and QMI, and pass those handles in as parameters to this function. The second argument for QMI is optional. If no argument is passed in for the second

QMI argument, QUTS selects a default QMI protocol. The logic for selecting the default QMI protocol is to find one without "MUX" in the description name. If more than one such QMI protocol is present, QUTS will select any one. If only "MUX" named protocols are present, QUTS will select any one. To use specific QMI protocol, pass in the protocol as second argument.

### Prototype

```
Common.ErrorCode initializeServiceByProtocol(1:i64 diagProtocolHandle,  
                                           2:i64 qmiProtocolHandle)
```

### Parameters

diagProtocolHandle	Specific diag protocol to use for the service.
qmiProtocolHandle	Specific QMI protocol to use for the service.

### Returns

None.

#### 5.1.3.5 destroyService()

Ends this instance of a service.

### Prototype

```
Common.ErrorCode destroyService()
```

#### 5.1.3.6 backupToXqcn()

Collects and saves configured NV items and EFS files into a XQCN file.

### Prototype

```
string backupToXqcn(1:string serviceProgrammingCode,  
                   2:bool resetUponCompletion,  
                   3:i32 resetTimeout,  
                   4:string filter)
```

### Parameters

serviceProgrammingCode	Six digit SPC for security access to the device.
resetUponCompletion	Whether to reset the device to normal operating mode after restoring.
resetTimeout	Time in milliseconds to wait for the device to reset if resetUponCompletion is true
filter	Optional filter for which items to restore. This parameter is empty if it is not needed.

## Returns

Contents of the XQCN file.

### 5.1.3.7 restoreFromXqcn()

Sets all NV items and EFS files on a device based on data from a specified XQCN file.

## Prototype

```
Common.ErrorCode restoreFromXqcn(1:string xqcnFileContents,  
                                  2:string serviceProgrammingCode,  
                                  3:bool allowEsnMismatch,  
                                  4:bool resetUponCompletion,  
                                  5:i32 resetTimeout,  
                                  6:string filterFileContents)
```

## Parameters

xqcnFileContents	Full contents of the XQCN file to use for restore.
serviceProgrammingCode	Six digit SPC for security access to the device.
allowEsnMismatch	Indicates that the operation can proceed if the device ESN does not match the ESN of the XQCN file.
resetUponCompletion	Whether to reset the device to normal operating mode after restoring.
resetTimeout	Time (in milliseconds) to wait for the device to reset if resetUponCompletion is true
filterFileContents	Optional filter for which items to restore. This parameter is empty if it is not needed.

## Returns

None.

### 5.1.3.8 pdcGetMbnMaxStorage()

Checks the MBN storage capacity for a specified MBN type.

## Prototype

```
i64 pdcGetMbnMaxStorage(1:PdcMbnType mbnType)
```

## Parameters

mbnType	Type of configuration memory store on which to operate ( <a href="#">PdcMbnType</a> ).
---------	--

## Returns

Storage capacity in number of bytes.

#### 5.1.3.9 pdcGetMbnStorageUsage()

Checks the MBN storage used for a specified MBN type.

##### Prototype

```
i64 pdcGetMbnStorageUsage(1:PdcMbnType mbnType)
```

##### Parameters

mbnType	Type of configuration memory store on which to operate ( <a href="#">PdcMbnType</a> ).
---------	--

##### Returns

Storage used in number of bytes.

#### 5.1.3.10 pdcGetMbnCount()

Checks the MBN count for a specified MBN type.

##### Prototype

```
i64 pdcGetMbnCount(1:PdcMbnType mbnType)
```

##### Parameters

mbnType	Type of configuration memory store on which to operate ( <a href="#">PdcMbnType</a> ).
---------	--

##### Returns

Number of MBNs.

#### 5.1.3.11 pdcGetMbnList()

Retrieves information of all MBNs for a specified MBN type.

##### Prototype

```
list<PdcMbnInfo> pdcGetMbnList(1:PdcMbnType mbnType)
```

##### Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
---------	--

##### Returns

List of MBN properties.

#### 5.1.3.12 pdcGetDefaultMbnInfo()

Retrieves embedded MBN information for a specified MBN type.

##### Prototype

```
PdcMbnInfo pdcGetDefaultMbnInfo(1:PdcMbnType mbnType)
```

##### Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
---------	--

##### Returns

Properties of the embedded MBN.

#### 5.1.3.13 pdcGetMbnId()

Calculate MBN ID from a MBN configuration.

##### Prototype

```
binary pdcReadMbn(1:binary mbnContent)
```

##### Parameters

mbnContent	MBN payload buffer.
------------	---------------------

##### Returns

20-byte MBN ID.

#### 5.1.3.14 pdcWriteMbn()

Loads an MBN to a device.

## Prototype

```
Common.ErrorCode pdcWriteMbn(1:PdcMbnType mbnType,  
                             2:binary mbnContent)
```

## Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
mbnContent	MBN payload buffer.

## Returns

None.

### 5.1.3.15 pdcReadMbn()

Reads the MBN payload.

## Prototype

```
binary pdcReadMbn(1:PdcMbnType mbnType,  
                  2:binary mbnId,  
                  3:i32 subId)
```

## Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
mbnId	20-byte MBN ID.
subId	Subscription ID to read the MBN from.

## Returns

Contents of an MBN stored in the device.

### 5.1.3.16 pdcRemoveMbn()

Deletes an MBN from a device.

## Prototype

```
Common.ErrorCode pdcRemoveMbn(1:PdcMbnType mbnType,  
                               2:binary mbnId)
```

## Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
mbnId	20-byte MBN ID.



### 5.1.3.17 pdcGetActiveMbn()

Checks the current MBN in use.

#### Prototype

```
binary pdcGetActiveMbn(1:PdcMbnType mbnType,  
                       2:i32 subId,  
                       3:i32 slotId)
```

#### Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
subId	Subscription ID to check active MBN.
slotId	Slot ID to check the active MBN.

#### Returns

20-byte MBN ID.

### 5.1.3.18 pdcGetPendingMbn()

Checks the specified MBN to be used next.

#### Prototype

```
binary pdcGetPendingMbn(1:PdcMbnType mbnType,  
                        2:i32 subId,  
                        3:i32 slotId)
```

#### Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
subId	Subscription ID to check pending MBN.
slotId	Slot ID to check pending MBN.

#### Returns

20-byte MBN ID.

#### 5.1.3.19 pdcValidateMbn()

Validates an MBN for a specified subscription.

##### Prototype

```
PdcMbnResult pdcValidateMbn(1:PdcMbnType mbnType,  
                             2:binary mbnId,  
                             3:i32 subId,  
                             4:string remotePath)
```

##### Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
mbnId	20-byte MBN ID.
subId	Subscription ID to validate the MBN.
remotePath	File path in the remote file system.

##### Returns

Content of the MBN validation result.

#### 5.1.3.20 pdcSelectMbn()

Selects the next MBN to be used.

##### Prototype

```
Common.ErrorCode pdcSelectMbn(1:PdcMbnType mbnType,  
                               2:binary mbnId,  
                               3:i32 subId,  
                               4:i32 slotId)
```

##### Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
mbnId	20-byte MBN ID.
subId	Subscription ID to select the MBN.
slotId	Slot ID to select the MBN.

##### Returns

None.

#### 5.1.3.21 pdcActivateMbn()

Activates an MBN that was selected for a specified subscription.

## Prototype

```
Common.ErrorCode pdcActivateMbn(1:PdcMbnType mbnType,  
                                2:i32 subId,  
                                3:i32 slotId,  
                                4:i32 mode,  
                                5:i32 timeout)
```

## Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
subId	Subscription ID to activate the MBN.
slotId	Slot ID to activate the MBN.
mode	Activation type, <ul style="list-style-type: none"><li>0 – Regular activation for selected configuration(s)</li><li>1 – Refresh only without selecting any configuration</li></ul>
timeout	Maximum wait time in milliseconds until device is re-enumerated.

## Returns

None.

### 5.1.3.22 pdcDeactivateMbn()

Deactivates an active MBN being used for a specified subscription.

## Prototype

```
Common.ErrorCode pdcDeactivateMbn(1:PdcMbnType mbnType,  
                                   2:i32 subId)
```

## Parameters

mbnType	Type of configuration memory store to operate on ( <a href="#">PdcMbnType</a> ).
subId	Subscription ID to deactivate the MBN.

## Returns

None.

### 5.1.3.23 pdcGetConfiguration()

Gets the value of a configuration feature stored on device for the specified slot ID.

## Prototype

```
i32 pdcGetConfiguration(1:i32 slotId,  
                        2:PdcConfigurationType feature)
```

### Parameters

slotId	Slot ID from which to query configuration features.
feature	Type of configuration.

### Returns

Value of the configuration.

#### 5.1.3.24 pdcSetConfiguration()

Sets the value of a configuration feature stored on device for specified slot ID.

### Prototype

```
Common.ErrorCode pdcSetConfiguration(1:i32 slotId,  
                                     2:PdcConfigurationType feature,  
                                     3:i32 value)
```

### Parameters

slotId	Slot ID to set the configuration features.
feature	Type of configuration.
value	Value of the configuration feature.

### Returns

None.

#### 5.1.3.25 nvIsItemSupported()

Checks whether a specified NV item is configured to be available on a device.

### Prototype

```
bool nvIsItemSupported(1:string nvItemName,  
                       2:i32 subscriptionId)
```

### Parameters

nvItemName	EFS NV item name to check.
subscriptionId	SIM to check. Default: NO_SUBSCRIPTION_ID.

### Returns

TRUE when the NV item is configured on the device.

### 5.1.3.26 nvReadItem()

Reads an NV item from a device.

#### Prototype

```
NvData nvReadItem(1:string nvItemNameOrId,  
                  2:i32 subscriptionId,  
                  3:i8 index,  
                  4:NvReturns returnConfig)
```

#### Parameters

nvItemNameOrId	NV item name or ID. Either the NV ID or legacy name or the EFS path.
subscriptionId	SIM to read from. Default – NO_SUBSCRIPTION_ID.
index	Optional index for legacy NV Items. Generally this is not used, pass 0 by default.
returnConfig	Specifies what data to return based on <a href="#">NvReturnFlags</a> .

#### Returns

NV item (defined in [NvData](#)).

### 5.1.3.27 nvSetItem()

Sets an NV item on a device.

#### Prototype

```
Common.ErrorCode nvSetItem(1:string nvItemNameOrId,  
                             2:string valueList,  
                             3:i32 subscriptionId)
```

#### Parameters

nvItemNameOrId	NV item name or ID. Either the NV ID or legacy name or the EFS path.
valueList	Comma-separated values of each NV field (for example, "1,3,5,2,0") or a JSON.
subscriptionId	SIM to write to. Default – NO_SUBSCRIPTION_ID.

#### Returns

None.

### 5.1.3.28 nvGetAllItems ()

Returns all NVs with ID, name, description, and category.

## Prototype

```
list<Common.NvItem> nvGetAllItems()
```

## Parameters

None

## Returns

List of [NvItems](#).

### 5.1.3.29 nvGetItemDefintion ()

Returns the definition of the NV item from ID or name.

## Prototype

```
string nvGetItemDefintion(1:string nvItemNameOrId)
```

## Parameters

nvItemNameOrId	NV item name or ID: either the NV ID or legacy name.
----------------	--

## Returns

JSON containing the definition of the NvItem.

### 5.1.3.30 efsHasAlternateFileSystem()

Checks if a device supports the alternate file system.

## Prototype

```
bool efsHasAlternateFileSystem()
```

## Returns

TRUE when the alternate file system is available on the connected device.

### 5.1.3.31 efsCreateDirectory()

Creates a new directory on in the EFS of a device.

## Prototype

```
Common.ErrorCode efsCreateDirectory(1:string pathName,  
                                     2:FileSystem efsSystem)
```

## Parameters

pathName	Name of the directory to create.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

None.

### 5.1.3.32 efsRemoveDirectory()

Removes a directory from the EFS of a device.

NOTE: The directory must be empty to be removed. To remove a non-empty directory, use [efsRemoveTree\(\)](#).

## Prototype

```
Common.ErrorCode efsRemoveDirectory(1:string pathName,  
                                     2:FileSystem efsSystem)
```

## Parameters

pathName	Name of the directory to remove.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

None.

### 5.1.3.33 efsRemoveTree()

Removes a directory and any subdirectories and files contained within it.

## Prototype

```
Common.ErrorCode efsRemoveTree(1:string pathName,  
                                2:FileSystem efsSystem)
```

## Parameters

pathName	Name of the directory to remove.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

None.

#### 5.1.3.34 efsGetDirectoryContents()

Returns a list of items that exist in a specified directory.

##### Prototype

```
list<EfsItem> efsGetDirectoryContents(1:string pathName,  
                                     2:FileSystem efsSystem)
```

##### Parameters

pathName	Name of the directory to query.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

##### Returns

List of items in the directory. See [EfsItem](#) type.

#### 5.1.3.35 efsPutFile()

Copies a file into the EFS.

##### Prototype

```
Common.ErrorCode efsPutFile(1:string pathName,  
                             2:binary buffer,  
                             3:FileSystem efsSystem)
```

##### Parameters

pathName	Name of the file to write.
buffer	File contents to write.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

##### Returns

None.

#### 5.1.3.36 efsGetFile()

Copies a file from the EFS.

##### Prototype

```
binary efsGetFile(1:string pathName,  
                  2:FileSystem efsSystem)
```



### Parameters

pathName	Name of the file to read.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

### Returns

File contents of pathName.

### 5.1.3.37 efsDeleteFile()

Deletes a file from the EFS.

### Prototype

```
Common.ErrorCode efsDeleteFile(1:string pathName,  
                                2:FileSystem efsSystem)
```

### Parameters

pathName	Name of the file to delete.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

### Returns

None.

### 5.1.3.38 efsRename()

Renames a file or directory in the EFS.

### Prototype

```
Common.ErrorCode efsRename(1:string existingName,  
                             2:string newName,  
                             3:FileSystem efsSystem)
```

### Parameters

existingName	Name of the existing file or directory to rename.
newName	New name to assign to the file or directory.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

### Returns

None.

### 5.1.3.39 efsPathExists()

Checks whether a specified path is a valid location in the EFS.

#### Prototype

```
bool efsPathExists(1:string pathName,  
                  2:FileSystem efsSystem)
```

#### Parameters

pathName	Path to query.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

#### Returns

TRUE when the path exists on the EFS.

### 5.1.3.40 efsIsDirectory()

Checks whether a specified path is a valid directory in the EFS.

#### Prototype

```
bool efsIsDirectory(1:string pathName,  
                   2:FileSystem efsSystem)
```

#### Parameters

pathName	Name of the directory to query.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

#### Returns

TRUE when the path refers to an existing directory in the EFS.

### 5.1.3.41 efsIsFile()

Checks whether a specified path is a valid file in the EFS.

#### Prototype

```
bool efsIsFile(1:string pathName,  
              2:FileSystem efsSystem)
```

#### Parameters

pathName	Name of the file to query.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

TRUE when the path refers to an existing file on the EFS.

### 5.1.3.42 efsIsReady()

Returns TRUE if EFS is ready.

## Prototype

```
bool efsIsReady()
```

## Returns

TRUE when EFS is ready otherwise returns FALSE.

### 5.1.3.43 efsGetFileSize()

Gets the size in bytes for a specified file.

## Prototype

```
i32 efsGetFileSize(1:string pathName,  
                  2:FileSystem efsSystem)
```

## Parameters

pathName	Name of the file to query.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

Size in bytes of the file if it exists. Otherwise, 0.

### 5.1.3.44 efsGetFileChecksum()

Gets the MD5 checksum of a specified file from DIAG.

## Prototype

```
binary efsGetFileChecksum(1:string pathName,  
                           2:FileSystem efsSystem)
```

## Parameters

pathName	Name of the file to query.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

16-byte MD5 checksum value.

### 5.1.3.45 efsGetFileAttributes()

Gets the file attributes.

## Prototype

```
EfsFileAttributes efsGetFileAttributes(1:string pathName,  
                                       2:FileSystem efsSystem)
```

## Parameters

pathName	Name of the file to query.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

EFS file attributes (defined in [EfsFileAttributes](#)).

### 5.1.3.46 efsGetAvailableSpace()

Gets the EFS available space.

## Prototype

```
I64 efsGetAvailableSpace(1:FileSystem efsSystem)
```

## Parameters

efsSystem	Primary or alternate <a href="#">FileSystem</a> . Optional, default value is FS_PRIMARY.
-----------	--

## Returns

Available space in bytes.

### 5.1.3.47 forceEfsSync ()

Synchronize any pending EFS updates with storage media immediately.

Currently, sync is time consuming as it is done for whole file system ("/") on the device irrespective of the pathName string passed to it. However, pathName is still added to API to be in line with the legacy device diagnostics interface. If file or directory based partial sync is added to EFS functionality on the device in future, this interface need not change.

Note: There is an implicit (timer-based) sync which happens on the device upon update of any EFS item. EFS sync also happens upon graceful shutdown of the device. Back to back force sync operations can cause long delays in processing subsequent diagnostic commands and increased power consumption on the device among other side effects.

## Prototype

```
Common.ErrorCode forceEfsSync(1:string pathName,  
                             2:FileSystem efsSystem)
```

## Parameters

pathName	File or directory to sync. Please refer to the API description for more details.
efsSystem	Primary or alternate <a href="#">FileSystem</a> .

## Returns

None.

### 5.1.3.48 initializeServiceWithOptions()

Opens connections and starts a service. This is similar to `initializeServiceByProtocol` but this needs to be used if diag is to be started with some special options. Currently only a single option is supported (`enableHdlcOnly`), which keeps the device in HDLC only mode.

## Prototype

```
Common.ErrorCode initializeServiceWithOptions(  
1:DeviceConfigConnectionOptions connectionOptions)
```

## Parameters

connectionOptions	Additional connection options, see <a href="#">DeviceConfigConnectionOptions</a> .
-------------------	--

## Returns

None.

### 5.1.3.49 configureService()

Some service options can be set using this API.

## Prototype

```
Common.ErrorCode configureService(1:DeviceConfigOptions  
deviceConfigOptions) throws (1:Common.AppException e)
```

## Parameters

deviceConfigOptions	Additional options, see <a href="#">DeviceConfigOptions</a> .
---------------------	---

## Returns

None.

## 5.2 Diag service

The diag service provides basic diag I/O, including sending and receiving commands, setting the logging mask, and monitoring data using data queues. This service shares the diag protocol.

**Note:** Diag Service cannot be used to open DPL / ADPL port or logging. Use Raw Service for DPL / ADPL logging. Diag Service should be used only to set log mask and enable 0x11EB log packet.

## 5.2.1 Types

### 5.2.1.1 LogMaskFormat

Used to specify the format of a logging mask file in [setLoggingMask\(\)](#). The last three formats are same as the corresponding first three formats but they are used to tell QUTS to enable multisim for the corresponding (CFG/CFG2/DMC) format.

```
enum LogMaskFormat
{
    CFG_FORMAT = 0,
    CFG2_FORMAT,
    DMC_FORMAT,
    CFG_FORMAT_ENABLE_MULTISIM,
    CFG2_FORMAT_ENABLE_MULTISIM,
    DMC_FORMAT_ENABLE_MULTISIM
}
```

### 5.2.1.2 DiagConnectionOptions

Used to specify the connection options for diag service.

```
struct DiagConnectionOptions
{
    1: optional i64 protocolHandle;
    2: optional bool openQdssPort;
    3: optional bool enableDefaultQdssConfiguration;
    4: optional bool enableHdlcOnly
}
```

protocolHandle	Optional diag protocol handle. Protocol handle must be set if the device has more than one diag protocol. Use <code>DiagConnectionOptions.setprotocolHandle()</code>
openQdssPort	Optional parameter. Default set to true. Set this to false to manually connect to QDSS port. If set to true, QUTS automatically connects to QDSS port when diag service is created. By default, this is set to true.
enableDefaultQdssConfiguration	Optional parameter. Default set to true. Set this to false to manually configure QDSS. If set to true, QUTS automatically configure QDSS port when Diag Service is created. By default this is set to true. QUTS attempts to configure in the following priority order: 1. Diag over ATB 2. Diag over STM 3. Legacy diag
enableHdlcOnly	Optional parameter. Default set to false i.e. Quts will always try to switch the device to non-HDLC mode. If this is set to true, Quts will not switch the device to non-HDLC mode and device will remain in HDLC mode. Once set, this cannot be changed for the lifetime of the diag instance.

### 5.2.1.3 Subsystem

Subsystem to specify for health report log packets. See [resetHealthReportCounters](#) and [SetHealthReportTimer](#).

```
enum Subsystem
{
    MPSS      = 0x0832,
    APP       = 0x0229,
    LPASS     = 0x0E1F,
    SLPI      = 0x1A31,
    CDSP      = 0x2425,
    NPU       = 0x2623,
    AUDIO     = 0x0067,
    SENSOR    = 0x0068,
    WLAN      = 0x006B,
    CHARGER   = 0x006D
}
```

### 5.2.1.4 HealthReportConfigureStatus

Specifies the status from the responses to health report configure requests. See [resetHealthReportCounters](#) and [SetHealthReportTimer](#).

```
enum HealthReportConfigureStatus
{
    INVALID_STATUS      = -1,
    SUCCESS              = 0,
    UNABLE_TO_GENERATE_REQUESTED_HEALTH_REPORT = 1,
    INVALID_TIMER_UNIT   = 2,
    ERROR_SETTING_TIMER  = 3,
    TIMER_UPDATED_BUT_TIMER_NOT_ENABLED_DUE_TO_LOG_MASK = 4
}
```

## 5.2.2 Functions

### 5.2.2.1 getLastError()

Queries for the error from the last function call.

#### Prototype

```
Common.ErrorType getLastError()
```



## Returns

Error code and description.

### 5.2.2.2 `getDevice()`

Gets the device handle on which a service is created.

## Prototype

```
i64 getDevice()
```

## Returns

Device handle.

### 5.2.2.3 `initializeService()`

Opens connections and starts the service.

**NOTE:** If more than one DIAG protocol is available on the device, this function fails. For such devices, use [initializeServiceByProtocol\(\)](#) instead.

## Prototype

```
Common.ErrorCode initializeService()
```

## Returns

None.

### 5.2.2.4 `initializeServiceByProtocol()`

Opens connections and starts a service. In the case that a device has multiple diag connections (such as fusion devices), this API must be used instead of [initializeService\(\)](#). Call [getProtocolList\(\)](#) on the device that is to be connected, determine the appropriate instance of diag, and pass its handle in as a parameter to this function.

## Prototype

```
Common.ErrorCode initializeServiceByProtocol(i64 protocolHandle)
```

## Parameters

<code>protocolHandle</code>	Protocol to use for the service.
-----------------------------	----------------------------------

## Returns

None.

### 5.2.2.5 destroyService()

Ends this instance of a service.

#### Prototype

```
Common.ErrorCode destroyService()
```

#### Returns

None.

### 5.2.2.6 sendRawRequest()

Sends a request over a specified protocol.

#### Prototype

```
Common.DiagPacket sendRawRequest(1:binary request,  
                                2:Common.DiagReturns returnConfig,  
                                3:i32 timeout)
```

#### Parameters

request	Raw buffer of the request.
returnConfig	Specifies what to set in <a href="#">DiagReturns</a> .
timeout	Timeout when to stop waiting for the response (in ms). Default set it to -1.

#### Detailed description

The input is a raw buffer expected to be in the format described by the DIAG ICD.

This function will block until the response is received. For non-blocking, use [sendRawRequestAsync\(\)](#).

#### Returns

[DiagPacket](#) structure.

Only PACKET\_NAME, BINARY\_PAYLOAD and PARSED\_TEXT ([DiagReturnFlags](#)) are valid for the return value.

### 5.2.2.7 sendRawRequestAsync()

Sends a request over a specified protocol.

The input is a raw buffer expected to be in the format described by the DIAG ICD.

#### Prototype

```
i64 sendRawRequestAsync(1:binary request)
```

## Parameters

request	Raw buffer of the request.
---------	----------------------------

## Returns

Transaction ID to use when querying the response using [getResponseAsync\(\)](#) or [getAllAsyncResponses\(\)](#).

### 5.2.2.8 sendRequest()

Sends a DIAG request based on a list of fields.

## Prototype

```
Common.DiagPacket sendRequest(1:Common.DiagPacketType requestType,  
                              2:string requestIdOrName,  
                              3:string fieldList,  
                              4:Common.DiagReturns returnConfig,  
                              5:i32 timeout)
```

## Parameters

requestType	Whether this is a normal request, subsystem request, or subsystem V2 request. See <a href="#">DiagPacketType</a> .
requestIdOrName	The ID or name of the request. See <a href="#">DiagPacketType</a> for details based on Common.DiagPacketType.
fieldList	Comma-delimited list of field values to put into the buffer.
returnConfig	Specifies what to set in <a href="#">DiagReturns</a> .
timeout	Timeout when to stop waiting for the response (in milliseconds). Default set it to -1.

## Detailed description

Using the commandCode, QUTS generates a buffer using the comma-separated values specified in fieldList. If fieldList does not contain enough fields to fully populate the request structure, all remaining fields are set to 0.

This function will block until the response is received. For non-blocking, use [sendRequestAsync\(\)](#).

## Returns

[DiagPacket](#) structure.

Only PACKET\_NAME, BINARY\_PAYLOAD, and PARSED\_TEXT ([DiagReturnFlags](#)) are valid for the return value.

### 5.2.2.9 sendRequestAsync()

Sends a diag request based on a list of fields.

#### Prototype

```
i64 sendRequestAsync(1:Common.DiagPacketType requestType,  
                    2:string requestIdOrName,  
                    3:string fieldList)
```

#### Parameters

requestType	Whether this is a normal request, subsystem request, or subsystem V2 request. See <a href="#">DiagPacketType</a> .
requestIdOrName	The ID or name of the request. See <a href="#">DiagPacketType</a> for details based on Common.DiagPacketType.
fieldList	Comma-delimited list of field values to put into the buffer.

#### Detailed description

Using the commandCode, QUTS generates a buffer using the comma-separated values specified in fieldList. If fieldList does not contain enough fields to fully populate the request structure, all remaining fields are set to 0.

#### Returns

Transaction ID to use when querying the response using [getResponseAsync\(\)](#) or [getAllAsyncResponses\(\)](#).

### 5.2.2.10 getResponseAsync()

Retrieves the response for a specified asynchronous request.

#### Prototype

```
Common.DiagPacket getResponseAsync(1:i64 transactionId,  
                                   2: Common.DiagReturns returnConfig,  
                                   3:i32 timeout)
```

#### Parameters

transactionId	Transaction received when calling a Send Asynchronous function.
returnConfig	Specifies what to set in <a href="#">DiagReturns</a> .
timeout	Timeout when to stop waiting for the response (in milliseconds).

## Detailed description

This function returns an empty buffer if the response is not received yet. Use [onAsyncResponse\(\)](#) to be notified when the asynchronous response is available.

If there are multiple responses for a specified transaction ID, [getResponseAsync\(\)](#) must be called multiple times. Each time, this function returns the next response in the order it was received.

## Returns

DiagPacket structure (see [DiagPacket](#)).

Only PACKET\_NAME, BINARY\_PAYLOAD, and PARSED\_TEXT ([DiagReturnFlags](#)) are valid for the return value.

When a response is successfully retrieved, the return value is cleared in QUTS.

### 5.2.2.11 getAllAsyncResponses()

Retrieves all responses for an asynchronous request that are received before the timeout.

## Prototype

```
list<Common.DiagPacket> getAllAsyncResponses(1:i64 transactionId,  
                                             2: Common.DiagReturns returnConfig,  
                                             3:i32 timeout)
```

## Parameters

transactionId	Transaction received when calling a Send Asynchronous function.
returnConfig	Specifies what to set in <a href="#">DiagReturns</a> .
timeout	Timeout when to stop waiting for the response (in milliseconds).

## Returns

List of DiagPacket (see [DiagPacket](#)) responses that corresponding to the request. The list is empty if no response is received before the timeout.

When a response is successfully retrieved, the returned values are cleared in QUTS.

### 5.2.2.12 createDataQueue()

Creates a queue that can be registered to incoming data packets for external processing.

This function is used in conjunction with protocol-specific register\*ForQueue functions and [getDataQueueItems\(\)](#).

## Prototype

```
Common.ErrorCode createDataQueue(1:string queueName,  
                                2:Common.DiagPacketFilter filter,  
                                3:Common.DiagReturnConfig returnConfig)
```

## Parameters

queueName	Unique name of the queue to be referenced from other functions.
filter	Determines what indications to include in this queue. See <a href="#">DiagPacketFilter</a> .
returnConfig	Determines what values to populate in returned DiagPackets. See <a href="#">DiagReturnConfig</a> .

## Returns

None.

### 5.2.2.13 removeDataQueue()

Removes a queue created from [createDataQueue\(\)](#).

## Prototype

```
Common.ErrorCode removeDataQueue(1:string queueName)
```

## Parameters

queueName	Unique name of the queue to remove.
-----------	-------------------------------------

## Returns

None.

### 5.2.2.14 clearDataQueue()

Removes all items from a specified data queue.

## Prototype

```
Common.ErrorCode clearDataQueue(1:string queueName)
```

## Parameters

queueName	Unique name of the queue to clear.
-----------	------------------------------------

## Returns

None.

#### 5.2.2.15 addDataQueueFilter()

Adds a filter to an existing data queue.

##### Prototype

```
Common.ErrorCode addDataQueueFilter(1:string queueName,  
                                     2:Common.DiagPacketFilter filter)
```

##### Parameters

queueName	Unique name of the queue to be referenced from other functions.
filter	Determines what indications to include in this queue. See <a href="#">DiagPacketFilter</a> .

##### Returns

None.

#### 5.2.2.16 removeDataQueueFilter()

Removes a filter from an existing data queue.

##### Prototype

```
Common.ErrorCode removeDataQueueFilter(1:string queueName,  
                                         2:Common.DiagPacketFilter filter)
```

##### Parameters

queueName	Unique name of the queue to be referenced from other functions.
filter	Determines what indications to include in this queue. See <a href="#">DiagPacketFilter</a> .

##### Returns

None.

#### 5.2.2.17 getDataQueueItems()

Gets items that have been accumulated in a queue based on the IDs for which the queue is registered.

##### Prototype

```
list<Common.DiagPacket> getDataQueueItems(1:string queueName,  
                                           2:i32 count,  
                                           3:i32 timeout)
```

### Parameters

queueName	Unique name of the queue.
count	Number of items to get from the queue.
timeout	Timeout when to stop waiting for a count of the items (in milliseconds).

### Returns

DiagPacket structure.

#### 5.2.2.18 setLoggingMask()

Sets a logging mask to be enabled on the diag connection from a specified CFG or DMC file.

### Prototype

```
Common.ErrorCode setLoggingMask(1:binary maskFileContent,  
                                2:LogMaskFormat format)
```

### Parameters

maskFileContent	Entire contents of the CFG or DMC file.
format	Whether the file is CFG, CFG2, or DMC

### Detailed description

CFG files are on-device logging configuration files that can be configured and generated in QXDM. These files can be used to configure the log, events, extended debug messages, and QTrace masks.

### Returns

None.

#### 5.2.2.19 getQShrinkState()

Returns the current QShrink state.

### Prototype

```
QShrink4DownloadState getQShrinkState()
```

### Returns

Current state of QShrink.

#### 5.2.2.20 loadQShrinkFile ()

Loads a QShrink file from given path.



## Prototype

```
void loadQShrinkFile(1. String path)
```

## Detailed description

The QShrink file specified in the path is loaded and the symbols from that Qshrink file are used.

## Returns

None.

### 5.2.2.21 setWcdmaProtocolRevision()

Set WCDMA protocol revision value from the values in the enum [WcdmaProtocolRevision](#).

## Prototype

```
Common.ErrorCode setWcdmaProtocolRevision(  
    1:WcdmaProtocolRevision revision)
```

## Detailed description

This function allows the user to set the WCDMA protocol revision that is used when parsing WCDMA OTA log items.

## Returns

None.

### 5.2.2.22 setCdmaProtocolRevision()

Set CDMA protocol revision value from the values in the enum [CdmaProtocolRevision](#).

## Prototype

```
Common.ErrorCode setCdmaProtocolRevision(  
    1:CdmaProtocolRevision revision)
```

## Detailed description

This function allows the user to set the CDMA protocol revision that is used when parsing CDMA OTA log items.

## Returns

None.

### 5.2.2.23 setPilotInc()

Sets the value of Pilot inc. The value is within the range (0:5).

#### Prototype

```
Common.ErrorCode setPilotInc(1:i16 pilotInc)
```

#### Returns

None.

### 5.2.2.24 initializeServiceWithOptions()

Opens connections and starts a service. In the case that a device has multiple diag connections (such as fusion devices), this API must be used instead of [initializeService\(\)](#). Call [getProtocolList\(\)](#) on the device that is to be connected, determine the appropriate instance of diag, and pass its handle in as a parameter to this function.

#### Prototype

```
Common.ErrorCode initializeServiceWithOptions(1:DiagConnectionOptions  
diagConnectionOptions)
```

#### Parameters

diagConnectionOptions	Additional connection options, see <a href="#">DiagConnectionOptions</a> .
-----------------------	--

#### Returns

None.

### 5.2.2.25 setLoggingMaskFromFilter()

Sets logging mask frm DiagPacketFilter.Prototype

```
Common.ErrorCode setLoggingMaskFromFilter(  
    1:Common.DiagPacketFilter filter,  
)
```

#### Parameters

filter	Determines what packets to include in this mask. See <a href="#">DiagPacketFilter</a> .
--------	---

### 5.2.2.26 resetHealthReportCounters()

Resets the health report counters and sequence number (for log packets 1xCD3 and 1xCDD). This is needed to be called before you start using the values in the health report packets.

### Prototype

HealthReportConfigureStatus resetHealthReportCounters(1:Subsystem subsystem)

### Parameters

subsystem	Determines what subsystem to reset the health counters and sequence number for. See <a href="#">Subsystem</a>
-----------	---

### Returns

Configuration reset response status. See [HealthReportConfigureStatus](#)

### 5.2.2.27 setHealthReportTimer ()

Sets the health report packets timer interval (for log packets 1xCD3 and 1xCDD). QUTS uses default timer of 15 seconds. This can be changed if needed via call to this method.

### Prototype

```
HealthReportConfigureStatus setHealthReportTimer(  
    1:Subsystem subsystem,  
    2:i32 timer)
```

### Parameters

subsystem	Determines what subsystem to set the health packet timer for. See <a href="#">Subsystem</a>
timer	Timer value in milliseconds

### Returns

Configuration timer response status. See [HealthReportConfigureStatus](#)

## 5.3 Image management service

The image management service provides the ability to read and write image files on the device for supporting build loading and memory dump collection. It uses the Sahara protocol for memory dump collection, and both Sahara and Firehose protocols for loading images to device. The Sahara and Firehose protocols will be locked so that it cannot be shared by other clients.

## 5.3.1 Types

### 5.3.1.1 DeviceImageMode

Used to specify the image transfer operation on device.

```
enum DeviceImageMode
{
    DEVICE_IMAGE_MODE_NONE = 0x00000000,
    DEVICE_IMAGE_MODE_SAHARA_DOWNLOAD = 0x00000001,
    DEVICE_IMAGE_MODE_SAHARA_CRASH = 0x00000002,
    DEVICE_IMAGE_MODE_SAHARA_EFS_SYNC = 0x00000004
}
```

### 5.3.1.2 MemoryType

Used to specify the memory storage type on device.

```
enum MemoryType
{
    MEMORY_TYPE_EMMC = 0,
    MEMORY_TYPE_UFS,
    MEMORY_TYPE_NAND,
    MEMORY_TYPE_SPINOR,
    MEMORY_TYPE_UNKNOWN
}
```

### 5.3.1.3 MemoryDumpStates

Specifies the state of the internal process while processing memory dump collection in [collectMemoryDump\(\)](#).

```
enum MemoryDumpStates
{
    INITIALIZE = 0,
    DOWNLOADING_TABLE,
    DOWNLOADING_PARTITION,
    DOWNLOADING_PARTITION_COMPLETE,
    DOWNLOADING_PROGRESS_REPORT,
    DOWNLOADING_DDR_DATA,
    RESETTING
}
```

#### 5.3.1.4 TransferImageResult

Returns results for an image transfer sequence.

```
struct TransferImageResult
{
    1: Common.ErrorCode errorCode;
    2: DeviceImageMode deviceImageMode;
}
```

#### 5.3.1.5 SaharaCommandResults

Returns consolidated device information from Sahara commands.

```
struct SaharaCommandResults {
    1: Common.ErrorCode errorCode;
    2: optional string serialNumber;
    3: optional string version
}
```

#### 5.3.1.6 DownloadBuildOptions

Specifies download options used by [downloadBuild](#).

```
struct DownloadBuildOptions {
    1: MemoryType          memoryType;
    2: optional bool       erase;
    3: optional list<string> rawXmlList;
    4: optional list<string> patchXmlList;
    5: optional string      firehoseProgPath;
    6: optional string      signedDigestsPath;
    7: optional string      chainedDigestsPath;
    8: optional bool        skipSahara;
    9: optional string      readImagesPath;
}
```

#### Fields

memoryType	Flash memory type used by Firehose protocol.
erase	Boolean flag indicating whether to erase the entire flash memory before programming.
rawXmlList	List of raw programmer xml file paths in a flattened build.
patchXmlList	List of patch programmer xml file paths in a flattened build.
firehoseProgPath	Firehose programmer file path.
signedDigestsPath	Signed digests image file path. Must be used together with chainedDigestsPath option.
chainedDigestsPath	Chained digests image file path. Must be used together with signedDigestsPath option.

skipSahara	Boolean flag indicating whether to go to Firehose directly without transferring device programmer using Sahara.
readImagesPath	Path to store images read from device. When this option is used, erase option will be ignored if specified, and <a href="#">downloadBuild</a> will read images from device instead of downloading build.

If both rawXmlList and patchXmlList are not specified, raw XML programmer files with filename containing "rawprogram" and patch XML programmer files with filename containing "patch" will be used as default xml programmers.

If either rawXmlList, or patchXmlList, or both are specified, only those specified XML programmers will be used in Firehose programming process.

## 5.3.2 Functions

### 5.3.2.1 getLastError()

Queries for the error from the last function call.

#### Prototype

```
Common.ErrorType getLastError()
```

#### Returns

Error code and description.

### 5.3.2.2 getDevice()

Gets the device handle on which this service is created.

#### Prototype

```
i64 getDevice()
```

#### Returns

Device handle.

### 5.3.2.3 initializeService()

Opens a Sahara connection and starts a service.

If more than one Sahara protocol is available, use [initializeServiceByProtocol\(\)](#) to initialize the service.

#### Prototype

```
Common.ErrorCode initializeService()
```

## Returns

None.

### 5.3.2.4 destroyService()

Ends this instance of a service.

## Prototype

```
Common.ErrorCode destroyService()
```

## Returns

None.

### 5.3.2.5 getDeviceImageMode()

Gets the expected image transfer operation on device.

## Prototype

```
DeviceImageMode getDeviceImageMode()
```

## Parameters

None.

## Returns

[DeviceImageMode\(\)](#) type.

### 5.3.2.6 getCommandResults ()

Collect results from all Sahara commands.

## Prototype

```
Common.ErrorCode getCommandResults(  
    1:SaharaCommandResults commandResults)
```

## Parameters

commandResults	Specifies what to set in <a href="#">SaharaCommandResults</a> .
----------------	---

## Returns

None.

### 5.3.2.7 transferImages ()

Send images to device that supports either flashless software loading or flash programmer loading using Sahara protocol.

#### Prototype

```
TransferImageResult transferImages(1:map<i32, string> imageList)
```

#### Parameters

imageList	An image file list containing image ID and image path of files to be transferred.
-----------	---

#### Returns

[TransferImageResult](#) structure. QUTS clients should check [DeviceImageMode](#) value in [TransferImageResult](#) to determine if a crash has happened on device that requires dump collection using the [collectMemoryDump\(\)](#) API.

### 5.3.2.8 downloadBuild ()

Download a flattened build to device that supports flash memory software loading using Firehose protocol.

#### Prototype

```
Common.ErrorCode downloadBuild(1:string buildPath,  
                                2:DownloadBuildOptions options)
```

#### Parameters

buildPath	Specifies flat build location.
options	Specifies download options, see <a href="#">DownloadBuildOptions</a> .

#### Returns

None.

### 5.3.2.9 collectMemoryDump()

Downloads all memory partitions from a device into a specified location in the event of a crash.

#### Prototype

```
Common.ErrorCode collectMemoryDump(1:string pathName)
```

#### Parameters

pathName	Folder in which to save the dumps.
----------	------------------------------------



### Detailed description

The state of the process is provided in the service event callback ([onServiceEvent\(\)](#)) using the following values (defined in [MemoryDumpStates](#)):

- `INITIALIZE` – Dump collection started.
- `DOWNLOADING_TABLE` – Partitions table is downloading. The count of partitions is provided in the description.
- `DOWNLOADING_PARTITION` – Memory partition is downloading. The name of the partition is provided in the description.
- `DOWNLOADING_PROGRESS_REPORT` – Report current download percentage. The percentage value is provided in the description.

### Returns

None.

#### 5.3.2.10 startRemoteEfsSync()

Start a process to syncing remote EFS files from a device into a specified location.

### Prototype

```
Common.ErrorCode startRemoteEfsSync(1:string pathName)
```

### Parameters

pathName	Folder in which to save the remote EFS files.
----------	---

### Detailed description

The syncing process will be running in the background continuously even after API returns. Call [stopRemoteEfsSync\(\)](#) to terminate the syncing process.

### Returns

None.

#### 5.3.2.11 stopRemoteEfsSync()

Stop the current remote EFS sync process.

### Prototype

```
Common.ErrorCode stopRemoteEfsSync()
```

### Parameters

None.

## Returns

None.

### 5.3.2.12 setDdrStorePath ()

Set path to store DDR training data.

## Prototype

```
Common.ErrorCode setDdrStorePath(1:string ddrStorePath)
```

## Parameters

ddrStorePath	Specifies path to store DDR training data.
--------------	--

## Returns

None.

### 5.3.2.13 resetDevice()

Resets a device to the normal operating mode after Firehose software download.

```
Common.ErrorCode resetDevice(1:i32 timeout)
```

## Parameters

timeout	Maximum wait time in seconds until device is re-enumerated.
---------	---

## Returns

The state of the process is provided in the service event callback ([onServiceEvent\(\)](#)) using the following values (defined in [MemoryDumpStates](#)):

- `RESETTING` – Device reset request has been sent to device.

### 5.3.2.14 switchToEdl()

Attempts to switch the device to Emergency download mode.

```
Common.ErrorCode switchToEdl()
```

## Parameters

None.

## Returns

None. QUTS clients can check the device mode through [getDeviceImageMode\(\)](#). Upon success, device image mode will be `DEVICE_IMAGE_MODE_SAHARA_DOWNLOAD`.

## 5.4 QMI service

The QMI service provides basic QMI I/O functionality, including sending and receiving commands and monitoring indications. This service shares the QMI protocol.

### 5.4.1 Functions

#### 5.4.1.1 getLastError()

Queries for the error from the last function call.

##### Prototype

```
Common.ErrorType getLastError()
```

##### Returns

Error code and description.

#### 5.4.1.2 getDevice()

Gets the device handle on which this service is created.

##### Prototype

```
i64 getDevice()
```

##### Returns

Device handle.

#### 5.4.1.3 initializeService()

Opens a QMI connection and starts a service.

If more than one QMI protocol is available, use [initializeServiceByProtocol\(\)](#) to initialize the service.

##### Prototype

```
Common.ErrorCode initializeService(1:string qmiServiceIdOrName)
```

##### Parameters

qmiServiceIdOrName	ID or name of the QMI connection on which to run this service.
--------------------	--

## Returns

None.

### 5.4.1.4 initializeServiceByProtocol()

Opens a QMI connection on a specified protocol and starts a service. In the case that a device has multiple QMI connections (such as fusion devices), this API must be used instead of [initializeService\(\)](#). Call [getProtocolList\(\)](#) on the device that is to be connected, determine the appropriate instance of QMI, and pass its handle in as a parameter to this function.

## Prototype

```
Common.ErrorCode initializeServiceByProtocol(1:i64 qmiProtocolHandle,  
                                           2:string qmiServiceIdOrName)
```

## Parameters

qmiProtocolHandle	QMI protocol to use for the service.
qmiServiceIdOrName	ID or name of the QMI service on which to run the service.

## Returns

None.

### 5.4.1.5 destroyService()

Ends this instance of a service.

## Prototype

```
Common.ErrorCode destroyService()
```

## Returns

None.

### 5.4.1.6 getQmiServiceId()

Returns the QMI service on which the connection is opened.

## Prototype

```
i32 getQmiServiceId()
```

## Returns

Service ID the QMI connection.

Upon an error, -1.

#### **5.4.1.7 sendRequest()**

Sends a QMI request based on an XML string.

##### **Prototype**

```
Common.QmiPacket sendRequest(1:string messageIdOrName,  
                             2:string xmlDefinition,  
                             3:Common.QmiReturnFlags returnFlags,  
                             4:i32 timeout)
```

## Parameters

messageIdOrName	QMI message ID or name for which to make the buffer.
xmlDefinition	XML string that defines the values to set for the fields in the request.
returnConfig	Specifies what to set in <a href="#">QmiReturns</a> .
timeout	Timeout when to stop waiting for the response (in ms).

## Detailed description

Using the service ID of the connection and the specified message ID, QUTS generates a buffer using the XML string. Any missing fields are assigned the value of 0.

This function will block until the response is received. For non-blocking, use [sendRequestAsync\(\)](#).

## Returns

Common.QmiPacket structure ([QmiPacket](#)).

### 5.4.1.8 sendRequestAsync()

Sends a QMI request based on an XML string.

## Prototype

```
i64 sendQmiRequestAsync(1:string messageIdOrName,  
                        2:string xmlDefinition)
```

## Parameters

messageIdOrName	QMI message ID or name for which to make the buffer.
xmlDefinition	XML string that contains the values to set for the fields in the request.

## Detailed description

Using the service ID of the connection and the specified message ID, QUTS generates a buffer using the XML string. Any missing fields are assigned the value of 0.

## Returns

Transaction ID to use when querying the response for this request via [getResponseAsync\(\)](#).

#### 5.4.1.9 sendRawRequest()

Sends a QMI request based on a raw payload.

##### Prototype

```
CommonQmiPacket sendRequest(1: binary request,  
                             2:Common.QmiReturns returnConfig,  
                             3:i32 timeout)
```

##### Parameters

request	Raw buffer of the request.
returnConfig	Specifies what to set in <a href="#">QmiReturns</a> .
timeout	Timeout when to stop waiting for the response (in ms).

##### Detailed description

Using the service ID of the connection and the QUTS will send the request as provided in the request parameter.

This function will block until the response is received. For non-blocking, use [sendRequestAsync\(\)](#).

##### Returns

Common.QmiPacket structure ([QmiPacket](#)).

#### 5.4.1.10 sendRawRequestAsync()

Sends a QMI request based on a raw payload.

##### Prototype

```
i64 sendQmiRequestAsync(1: binary request)
```

##### Parameters

request	Raw buffer of the request.
---------	----------------------------

##### Detailed description

Using the service ID of the connection and the specified message ID, QUTS generates a buffer using the XML string. Any missing fields are assigned the value of 0.

##### Returns

Transaction ID to use when querying the response for this request via [getResponseAsync\(\)](#).

#### 5.4.1.11 `getResponseAsync()`

Retrieves the response for a specified asynchronous request.

##### Prototype

```
Common.QmiPacket getResponseAsync(1:i64 transactionId,  
                                   2:Common.QmiReturns returnConfig,  
                                   3:i32 timeout)
```

##### Parameters

transactionId	Transaction received when calling a <a href="#">sendRequestAsync()</a> function.
returnConfig	Specifies what to set in <a href="#">QmiReturns</a> .
timeout	Timeout when to stop waiting for the response (in milliseconds).

##### Detailed description

This function returns an empty buffer if the response is not received yet. Use [onAsyncResponse\(\)](#) to notify this function when the asynchronous response is available.

##### Returns

Common.QmiPacket structure ([QmiPacket](#)).

On successful retrieval for a response, the returned value is cleared in QUTS.

#### 5.4.1.12 `createIndicationQueue()`

Creates a queue that can be registered to incoming data packets for external processing.

This function is used in conjunction with [getIndications\(\)](#).

##### Prototype

```
Common.ErrorCode createDataQueue(1:string queueName  
                                 2:Common.QmiPacketFilter filter,  
                                 3:Common.QmiReturnConfig returnConfig)
```

##### Parameters

queueName	Unique name of the queue for referencing it from other functions.
filter	Determines what indications to include in this queue. See <a href="#">QmiPacketFilter</a> .
returnConfig	Determines what values to populate in returned QmiPackets. See <a href="#">QmiReturnConfig</a> .

##### Returns

None.



#### 5.4.1.13 removeIndicationQueue()

Removes a queue created from [createIndicationQueue\(\)](#).

##### Prototype

```
Common.ErrorCode removeDataQueue(1:string queueName)
```

##### Parameters

queueName	Unique name of the queue.
-----------	---------------------------

##### Returns

None.

#### 5.4.1.14 clearIndicationQueue()

Removes all items from a specified queue.

##### Prototype

```
Common.ErrorCode clearIndicationQueue(1:string queueName)
```

##### Parameters

queueName	Unique name of the queue.
-----------	---------------------------

##### Returns

None.

#### 5.4.1.15 getIndications()

Gets the indications that have been accumulated in a queue based on the IDs for which the queue is registered.

##### Prototype

```
list<Common.QmiPacket> getIndications(1:string queueName,  
                                     2:i32 count,  
                                     3:i32 timeout)
```

##### Parameters

queueName	Unique name of the queue.
count	Number of indications to get from the queue.
timeout	Timeout when to stop waiting for a count of the indications (in ms).

Returns

Common.QmiPacket structure ([QmiPacket](#)).

5.5 Raw service

Raw service provides basic I/O functionality, including sending and receiving commands. It is intended for features that are under development and do not yet have official support. It is not recommended to use raw service for any other purposes. Use this service to enable DPL / ADPL logging on the device.

DPL / ADPL logging using Raw Service:

To enable adpl logging use Quts Raw Service. Pass the DPL / ADPL protocol handle to initialize this service and connect to it.

Quts supports loading “.adpl , .adplv2, .adpl v3”, .adplv4 files. Use Quts Log Session to load the files.

QXDM5 + Quts is verified for ADPL log collection.

Diag Service is not used to start DPL / ADPL logging. Diag Service should be used only to load DMC with 0x11EB packets.

5.5.1 Constants

5.5.2 Types

5.5.2.1 QdssConfiguration

QDSS configuration options for raw service (see [ConnectionOptions](#)).

```
struct QdssConfiguration
{
    1: list<i16> stpv2Atids;
    2: list<i16> stmAtids;
    3: optional list<i16> diagEntityId;
}
```

stpv2Atids	List of AT IDs that use the stpv2.protocol
stmAtids	List of AT IDs for stm.
diagEntityId	List of diag entities.

5.5.2.2 ConnectionOptions

Connection options for raw service to configure different protocols.

```
struct ConnectionOptions
{
    1: optional QdssConfiguration qdssConfiguration;
```

}

qdssConfiguration	Set <a href="#">QdssConfiguration</a> options.
-------------------	--

## 5.5.3 Functions

### 5.5.3.1 getLastError()

Queries for the error from the last function call.

#### Prototype

```
Common.ErrorType getLastError()
```

#### Returns

Error code and description.

### 5.5.3.2 getDevice()

Gets the device to which this service instance is attached.

#### Prototype

```
i64 getDevice()
```

#### Returns

Device handle.

### 5.5.3.3 initializeService()

Creates a connection to a device over a specified protocol.

#### Prototype

```
Common.ErrorCode initializeService(1:i64 protocolHandle,  
                                   2:Common.OpenProp access,  
                                   3:Common.OpenProp share)
```

#### Parameters

protocolHandle	Protocol to open.
access	Open with read or read/write access.
share	Allow sharing the read or read/write access.

#### Returns

None.

#### 5.5.3.4 initializeServiceQmi()

Creates a QMI connection to a device over a specified protocol and service ID.

##### Prototype

```
Common.ErrorCode initializeServiceQmi(1:i64 protocolHandle,  
                                     2:i32 serviceId,  
                                     3:Common.OpenProp access,  
                                     4:Common.OpenProp share)
```

##### Parameters

protocolHandle	Protocol to open.
serviceId	ID of the QMI service.
access	Open with read or read/write access.
share	Allow sharing the read or read/write access.

##### Returns

None.

#### 5.5.3.5 destroyService()

Closes a specified connection to a device and destroys this instance of the raw service.

##### Prototype

```
Common.ErrorCode destroyService()
```

##### Returns

None.

#### 5.5.3.6 sendRequest()

Sends a request over a specified protocol.

##### Prototype

```
binary sendRequest(1:binary request,  
                  2:i32 timeout)
```

##### Parameters

request	Binary request payload to send
timeout	Timeout when to stop waiting for the response (in milliseconds).

### Detailed description

The input is a raw buffer expected to be in the format expected by the protocol that was opened in this service in [initializeService\(\)](#) or [initializeServiceQmi\(\)](#).

This function will block until the response is received or timed out. For non-blocking, use [sendRequestAsync\(\)](#).

### Returns

Response that corresponds to the request. Upon an error, this return value is empty.

#### 5.5.3.7 sendRequestAsync()

Sends a request over the specified protocol.

### Prototype

```
i64 sendRequestAsync(1:binary request)
```

### Parameters

request	Raw buffer of the request.
---------	----------------------------

### Detailed description

The input is a raw buffer expected to be in the format expected by the protocol that was opened in this service in [initializeService\(\)](#) or [initializeServiceQmi\(\)](#).

### Returns

Transaction ID to use when querying the response for this request using [getResponseAsync\(\)](#) or [getAllResponsesAsync\(\)](#).

#### 5.5.3.8 getResponseAsync()

Retrieves the response for a specified asynchronous request.

### Prototype

```
binary getResponseAsync(1:i64 transactionId,  
                        2:i32 timeout)
```

### Parameters

transactionId	Transaction received when calling a <a href="#">sendRequestAsync()</a> function.
timeout	Timeout when to stop waiting for the response (in milliseconds).

### Detailed description

This function returns an empty buffer if the response is not received yet. Use [onAsyncResponse\(\)](#) to notify this function when the asynchronous response is available.

If a specified transaction ID has multiple responses, call [getResponseAsync\(\)](#) multiple times. Each time, the function will return the next response in the order it was received. Alternatively, call [getAllResponsesAsync\(\)](#) to retrieve a set of responses.

### **Returns**

Response corresponding to the request. This return is empty if the response has not been received.

If a response is successfully retrieved, the returned value is cleared in QUTS.

### **5.5.3.9 getAllResponsesAsync()**

Retrieves all responses for an asynchronous request if they are received before the timeout.

### **Prototype**

```
list<binary> getAllResponsesAsync(1:i64 transactionId,  
                                2:i32 timeout)
```

### **Parameters**

transactionId	Transaction received when calling a <a href="#">sendRequestAsync</a> function.
timeout	Timeout when to stop waiting for the response (in milliseconds).

### **Returns**

List of responses corresponding to the request. This list is empty if no response is received before the timeout.

If a response is successfully retrieved, the returned value is cleared in QUTS.

### **5.5.3.10 isAsyncResponseFinished()**

Checks whether any more responses are expected for an asynchronous request.

### **Prototype**

```
bool isAsyncResponseFinished(1:i64 transactionId)
```

### **Parameters**

transactionId	Transaction to check.
---------------	-----------------------

### **Returns**

TRUE when no further responses are expected for the request.

#### 5.5.3.11 initializeServiceWithOptions()

Creates a QMI connection to a device over a specified protocol and service ID.

##### Prototype

```
Common.ErrorCode initializeServiceWithOptions(1:i64 protocolHandle,  
                                              2:Common.OpenProp access,  
                                              3:Common.OpenProp share,  
                                              4:ConnectionOptions  
connectionOptions)
```

##### Parameters

protocolHandle	Protocol to open.
access	Open with read or read/write access.
share	Allow sharing the read or read/write access.
connectionOptions	Configure connection options for different protocols see <a href="#">ConnectionOptions</a>

##### Returns

None.

## 5.6 ADB service

The ADB service provides basic ADB I/O functionality, including sending and receiving commands.

### 5.6.1 Functions

#### 5.6.1.1 getLastError()

Queries for the error from the last function call.

##### Prototype

```
Common.ErrorType getLastError()
```

##### Returns

Error code and description.

#### 5.6.1.2 getDevice()

Gets the device to which this service instance is attached.

##### Prototype

```
i64 getDevice()
```

## Returns

Device handle.

### 5.6.1.3 initializeService()

Creates a connection to a device over a ADB protocol if only one ADB protocol is present. If more than one ADB protocol is present, this will throw an exception. Use [initializeServiceByProtocol\(\)](#) in that case by passing in the specific ADB protocol of interest.

## Prototype

```
Common.ErrorCode initializeService(1:Common.OpenProp access,  
                                   2:Common.OpenProp share)
```

## Parameters

access	Open with read or read/write access.
share	Allow sharing the read or read/write access.

## Returns

None.

### 5.6.1.4 destroyService()

Closes a specified connection to a device and destroys this instance of the ADB service.

## Prototype

```
Common.ErrorCode destroyService()
```

## Returns

None.

### 5.6.1.5 sendCommand()

Sends an ADB command. API is blocking until response is received.

## Prototype

```
Common.AdbPacket sendCommand(  
1:string command,  
2:Common.AdbReturnConfig returnConfig,  
3:i32 timeout  
)
```

## Parameters



---

***Error! No text of specified style in document.***

command	ADB request string to send.
returnConfig	returnConfig of type <a href="#">AdbReturnConfig</a> with flags set to determine fields in return value.
timeout	Timeout when to stop waiting for the response (in ms).

### Detailed description

The input is an ADB string (for example, "adb devices").

This function will block until the response is received or timed out. For non-blocking, use [sendCommandAsync\(\)](#).

### Returns

[AdbPacket](#) object containing the response that corresponds to the ADB command. Fields in this return value are determined by flags set in returnConfig parameter.

#### 5.6.1.6 sendCommandAsync()

Sends an ADB command asynchronously.

### Prototype

```
i64 sendCommandAsync(1:string command)
```

### Parameters

command	Adb string of the request.
---------	----------------------------

### Detailed description

The input is an ADB string (for example, "adb devices").

### Returns

Transaction ID to use when querying the response for this request using [getResponseAsync\(\)](#) or [getAllResponsesAsync\(\)](#).

#### 5.6.1.7 getResponseAsync()

Retrieves the response for a specified asynchronous request.

### Prototype

```
Common.AdbPacket getResponseAsync(  
1:i64 transactionId,  
2:Common.AdbReturnConfig returnConfig,  
3:i32 timeout)
```

### Parameters

transactionId	Transaction received when calling a <a href="#">sendRequestAsync</a> function.
returnConfig	returnConfig of type <a href="#">AdbReturnConfig</a> with flags set to determine fields in return value.
timeout	Timeout when to stop waiting for the response (in ms).

## Detailed description

This function returns an empty buffer if the response is not received yet. Use [onAsyncResponse\(\)](#) to notify this function when the asynchronous response is available.

If a specified transaction ID has multiple responses, call [getResponseAsync\(\)](#) multiple times. Each time, the function will return the next response in the order it was received.

Alternatively, call [getAllResponsesAsync\(\)](#) to retrieve a set of responses.

## Returns

[AdbPacket](#) object containing the response that corresponds to the ADB command. Fields in this return value are determined by flags set in the returnConfig parameter.

If a response is successfully retrieved, the returned value is cleared in QUTS.

### 5.6.1.8 getAllResponsesAsync()

Retrieves all responses for an asynchronous request if they are received before the timeout.

## Prototype

```
list<Common.AdbPacket> getAllResponsesAsync(  
1:i64 transactionId,  
2:Common.AdbReturnConfig returnConfig,  
3:i32 timeout)
```

## Parameters

transactionId	Transaction received when calling a Send Asynchronous function.
returnConfig	returnConfig of type <a href="#">AdbReturnConfig</a> with flags set to determine fields in return value.
timeout	Timeout when to stop waiting for the response (in ms).

## Returns

List of [AdbPacket](#) objects containing responses that corresponds to the ADB command. Fields in each AdbPacket are determined by flags set in returnConfig parameter.

If a response is successfully retrieved, the returned value is cleared in QUTS.

#### 5.6.1.9 isAsyncResponseFinished()

Checks whether any more responses are expected for an asynchronous request.

##### Prototype

```
bool isAsyncResponseFinished(1:i64 transactionId)
```

##### Parameters

transactionId	Transaction to check.
---------------	-----------------------

##### Returns

TRUE when no further responses are expected for the request.

## 5.7 GPS service

The GPS service provides basic GPS functionality `getCurrentLocation()` for NMEA protocol based devices.

#### 5.7.1.1 Note 1:

Currently QUTS does not have a way to automatically detect if the protocol shown up by NMEA/GPS device is a NMEA protocol, so it will show up as unknown protocol in protocol list. So once you get the protocolHandle (say via call to `getProtocolList`) which corresponds to NMEA that you can identify based on the specific GPS device name, you need to tell QUTS that it is a NMEA protocol by calling `overrideUnknownProtocol()`. This needs to be done before call to `createService()` on GPS. For e.g., in python the call would look like:

```
result = deviceManager.overrideUnknownProtocol(  
    unknownProtocolHandle,  
    Common.ttypes.ProtocolType.PROT_NMEA)  
  
gpsService = GpsService.GpsService.Client(  
    qutsClient.CreateService("GPS Service", deviceId))
```

#### 5.7.1.2 Note 2:

Quts currently uses baud rate of 4800 to talk to device over USB. If the NMEA device doesn't show up in the QUTS protocol list automatically, try changing the baud rate (bits per second) of the device from windows settings (Device Manager -> Ports (COM & LPT) -> Double click the NMEA device among list -> Port Settings Tab -> change the Bits per second to 4800) and retry.

## 5.7.2 Functions

### 5.7.2.1 getLastError()

Queries for the error from the last function call.

#### Prototype

```
Common.ErrorType getLastError()
```

#### Returns

Error code and description.

### 5.7.2.2 getDevice()

Gets the device to which this service instance is attached.

#### Prototype

```
i64 getDevice()
```

#### Returns

Device handle.

### 5.7.2.3 initializeService()

Creates a connection to a device over NMEA protocol if only one NMEA protocol is present. If more than one NMEA protocol is present, this will throw an exception. Use `initializeServiceByProtocol()` in that case by passing in the specific NMEA protocol of interest. See [Note1](#) and [Note2](#).

#### Prototype

```
Common.ErrorCode initializeService(1:Common.OpenProp access,  
                                   2:Common.OpenProp share)
```

#### Parameters

access	Open with read or read/write access.
share	Allow sharing the read or read/write access.

#### Returns

None.

#### 5.7.2.4 initializeServiceByProtocol()

Opens connections and starts a service. In the case that a device has multiple nmea connections or multiple NMEA devices are attached, this API must be used instead of [initializeService\(\)](#). Call [getProtocolList\(\)](#) on the device that is to be connected, determine the appropriate instance of nmea, and pass its handle in as a parameter to this function.

See [Note1](#) and [Note2](#).

##### Prototype

```
Common.ErrorCode initializeServiceByProtocol(1:i64 protocolHandle)
```

##### Parameters

protocolHandle	Protocol to use for the service.
----------------	----------------------------------

##### Returns

None.

#### 5.7.2.5 destroyService()

Closes a specified connection to a device and destroys this instance of the GPS service.

##### Prototype

```
Common.ErrorCode destroyService()
```

##### Returns

None.

#### 5.7.2.6 getCurrentLocation()

Returns the current location as determined by the GPS device.

##### Prototype

```
Common.GpsPacket getCurrentLocation()
```

##### Parameters

None

##### Returns

[GpsPacket](#) object containing the GPS details such as location, speed etc.

## A References

### 5.7.3 Acronyms and terms

Acronym or term	Definition
EFS	Encrypting File System
ESN	Electronic Serial Number
MBN	MCFG binaries
NV	Non-volatile
QDSS	Qualcomm Debug Subsystem
QMI	Qualcomm MSM Interface
QSH	Qualcomm Sherlock Holmes
QUTS	Qualcomm Unified Tools Service
RPC	Remote Procedure Call
SPC	Service Programming Code