Qualcomm Technologies, Inc.

Qualcomm Unified Tools Service (QUTS) Scripting Guide

80-PG522-2 Rev. A

July 23, 2019

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

Revision history

Revision	Date	Description
A	July 2018	Initial release

Contents

1 Introduction

1.1 Purpose

This document describes how to use QUTS APIs for automation. QXDM Professional APIs will no longer be used and QXDM no longer needs to be running in order for automation scripts to work. QXDM will now only be used when the user wants to visually see the data. There is no one-to one-mapping of APIs between QUTS and QXDM. For details on the API, refer to the QUTS users document installed in the C:\Program Files (x86)\Qualcomm\QUTS folder.

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:`**.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

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

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.

2 Starting QUTS

QUTS currently supports scripting using Python, C#, C++, Java and Perl. Python is used for samples in this document. QUTS installation includes supporting files for scripting in each language.

2.1 Add header files

In your script, include the following necessary header files:

```
import os
import sys
import time
import re

##The path where QUTS files are installed
sys.path.append('C:\Program Files (x86)\Qualcomm\QUTS\Support\python')
import QutsClient
import Common.ttypes
```

2.2 Create QUTS client

The following is an example of the creation of a QUTS client. The parameter input provides a name that can be used to identify the client in QUTS logs.

```
quts_client = QutsClient.QutsClient("QUTSSampleClient")
```

2.3 Create QUTS service

QUTS provides various services to handle different scenarios when using a device. Depending on the need, the user needs to create an instance of the service and start it using the functions supported.

- DeviceManager – Gets the details of the devices that are connected to the host machine. Details include device information, protocols supported, etc.
- DeviceConfigService – Supports functions to configure the device. Configuration include setting NV, MBN, etc.
- DiagService – Supports functions related to communicating with diag devices

- QMIService – Supports functions related to sending/receiving messages using QMI
- ADBService – Supports functions related to sending/receiving ADB commands
- LogSession – Supports functions related to collecting and postprocessing device data
- RawService – Supports functions to send/receive commands that are not yet supported in QUTS

The following is a typical call flow for communicating with a device and creating the appropriate service:

1. Call getServicesList() to get a list of available services and select 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.

2.4 Add import for diag access

The following import should be added for accessing diag. The import is different for each service.

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

    quts_client = QutsClient.QutsClient("LogSessionLiveSample")
##Initialize
    devManager = quts_client.getDeviceManager();

#Returns the list of devices that support Diag.
    deviceList =
devManager.getDevicesForService(DiagService.constants.DIAG_SERVICE_NAME);
    diagService =
DiagService.DiagService.Client(quts_client.createService(DiagService.consta
nts.DIAG_SERVICE_NAME, deviceList[0]))
    if(0 != diagService.initializeService()):
        print ("Diag init failed")
        return
```

2.5 Connect to device manager

Device manager provides device information and other commonly used functions such as phone restart, set/get operating mode, and data collection.

The following code snippet provides the handle to the device manager:

```
   quts_client = QutsClient.QutsClient("Sample")
   ##Get a handle to the device Manager. Device manager has all the device
related info.
    devManager = quts_client.getDeviceManager();
```

The device manager interface provides the details of the devices connected to the host. Users can get the list of devices connected. For each QTI device connected, users can also get additional information such as the protocols supported, serial number, location, and build info. For a complete list of the details provides, refer to the *Qualcomm Unified Tools Service (QUTS) API Guide* (80-PG522-1).

The following sample code connects to and gets device info from the device manager:

```
   quts_client = QutsClient.QutsClient("Sample")
   ##Get a handle to the device Manager. Device manager has all the device
related info.
    devManager = quts_client.getDeviceManager();

    deviceList =  devManager.getDeviceList(); ##gets the list of all the
devices connected to the system

    print ("Device Details", " Serial Number ", deviceList[0].serialNumber,
"  ", " Adb Serial Number ", deviceList[0].adbSerialNumber )
```

2.6 Configure the device

QUTS provides device configuration services to configure the device. This includes setting NV, EFS, and MBN. refer to the *Qualcomm Unified Tools Service (QUTS) API Guide* (80-PG522-1) for complete list of APIs.

The following is sample code to set an NV item:

```
quts_client = QutsClient.QutsClient("NV Sample")
   ##Get a handle to the device Manager. Device manager has all the device
related info.
    devManager = quts_client.getDeviceManager();

    deviceList =  devManager.getDeviceList(); ##gets the list of all the
devices connected to the system
```

```
    print ("Device Details", " Serial Number ", deviceList[0].serialNumber,
"  ", " Adb Serial Number ", deviceList[0].adbSerialNumber )


   ##get the list of Qualcomm supported devices
    devices =
quts_client.getDeviceManager().getDevicesForService( DeviceConfigService.co
nstants.DEVICE_CONFIG_SERVICE_NAME)
    devConfig =
DeviceConfigService.DeviceConfigService.Client(quts_client.createService(De
viceConfigService.constants.DEVICE_CONFIG_SERVICE_NAME, devices[0]))
    if(0 != devConfig.initializeService()):
       getLastError(devConfig)

   ##sets the NV item id 10 with value 21

   errorCode = devConfig.nvSetItem( "10",  "0,21", -1)
   if(errorCode != 0):
        print( "  Error in set NV " ,  devConfig.getLastError())
   else:
        print ("NV item set")
```

2.7 Configure Diag

QUTS provide users with this API to configure Diag:

```
::Uts::ErrorCode::type DeviceManagerHandler::configureProtocol(const
::Uts::ProtocolConfiguration& protocolConfiguration)
```

Here is a sample code:

```
quts_client = QutsClient.QutsClient("configure protocol")
   ##Get a handle to the device Manager. Device manager has all the device
related info.
    devManager = quts_client.getDeviceManager();

    protocol_config = Common.ttypes.ProtocolConfiguration()
    protocol_config.blockingLogMaskClearOnConnection = True
    protocol_config.protocolHandle = protocol_handle
    dev_manager.configureProtocol(protocol_config)
```

3 Collecting live data

Between QXDM and QUTS, there are some differences in the way devices are identified and connected. QUTS does not use a COM port number to identify the device. The user can get the list of devices (and device properties such as the name, serial number, build information, etc.) connected to the station as well as the list of the devices that support diag (or QMI, etc.) as follows:

```
   ##Get a handle to the device Manager. Device manager has all the device
related info.
    devManager = quts_client.getDeviceManager();


   ##Get the list of devices connected to the PC that supports Diag.
    deviceList =
devManager.getDevicesForService(DiagService.constants.DIAG_SERVICE_NAME);
```

All port related APIs in QXDM and QPST are no longer valid.

3.1 Connect to diag device

Once the list of the devices connected to the host has been obtained, connect to the intended device by initializing the diag service:

```
   ##Get the DiagService for the device. This handle will be used for
manipulating Diag (send/receive commands etc). The folloing line gets the
first Diag device connected in the PC.
    diagService =
DiagService.DiagService.Client(quts_client.createService(DiagService.consta
nts.DIAG_SERVICE_NAME, deviceList[0]))

   ##For a single modem device
    if(0 != diagService.initializeService()):
        print ("Diag init failed")
        return
```

The diagService variable is the handle to the diag service of the device selected. All interactions with the diag device are carried out using this handle.

When the interaction with the device is done, disconnect the device using the destroyService command.

```
diagService.destroyService()
```

Connecting to multiple diag devices in a single device

If a device has multiple diag devices (such as a fusion device), the diag service can be created based on the device and the protocol handle of the diag device. For each diag device, create the diag service for the device and provide the protocol handle of the diag device while initializing the service:

```
   ##For Fusion devices where there is multiple Diag devices, you need to
get the desired diag device's protocolHandle
    protList = devManager.getProtocolList(deviceList[0])
    listOfDiagServices = []
    for y in range(0, len(protList)  ):
        if(protList[y].protocolType ==
Common.ttypes.ProtocolType.PROT_DIAG )  :
                ##Get the DiagService for the device. This handle will be
used for manipulating Diag (send/receive commands etc). The following line
get the first   device connected in the PC.
            diagService =
DiagService.DiagService.Client(quts_client.createService(DiagService.consta
nts.DIAG_SERVICE_NAME, deviceList[0]))
            selectedProtocol = protList[y].protocolHandle;
            diagService.initializeServiceByProtocol(selectedProtocol);
            print("Using device , prot" ,
devManager.getChipName(deviceList[0], selectedProtocol))
             listOfDiagServices[y] = diagService
```

3.2 Set log mask

After connecting to the device, set the log mask as needed. To set the mask, use the contents of the CFG or DMC file saved from QXDM. Refer to the *Qualcomm Unified Tools Service (QUTS) User Guide* (80-PG522-3) for API details.

3.3 Send diag commands

Diag commands to the device can be sent using the sendRequest APIs of diagService. This is similar to the sendScript API of QXDM.

QUTS provides APIs for sending sync/async requests with immediate or delayed response. For more details on various ways of sending diag commands to the device, refer to the *Qualcomm Unified Tools Service (QUTS) User Guide* (80-PG522-3).

While sending diag commands, the diag packet type (request, subsys request, or subsysV2 request, etc.) must also to be sent. Also, the user needs to indicate which information from the response they are interested in (such as packet name, ID, parsed text, summary text, etc.).

```
diagPackettype = Common.ttypes.DiagPacketType.REQUEST

returnObj = Common.ttypes.DiagReturns()
returnObj.flags = Common.ttypes.DiagReturnFlags.RECEIVE_TIME_STRING    |
Common.ttypes.DiagReturnFlags.SUMMARY_TEXT |
Common.ttypes.DiagReturnFlags.PACKET_NAME |
Common.ttypes.DiagReturnFlags.PACKET_ID |
Common.ttypes.DiagReturnFlags.PACKET_TYPE
response= diagService.sendRequest(diagPackettype,"28","",returnObj,10000)
```

The response returned contains the information that is specified in the returnConfig parameter. A sample output for the above request is:

```
DiagPacket(errorCode=0, packetType=5, packetId='12',
moreResponsesFlag=None, sessionIndex=None, protocolIndex=None,
packetName='DMSS Status Response', timeStampData=None,
timeStampString=None, binaryPayload=None, parsedText=None,
subscriptionId=None, processorId=None, hwTimeStampData=None,
hwTimeStampString=None, ulogSource=None, receiveTimeData=None,
receiveTimeString=None, queryResultJson=None, summaryText=None,
transactionId=545)
```

The fields not mentioned in the returnConfig are set to none.

3.4 Collect data

QUTS allows users to collect data using the concept of DataQueues. Before starting to send diag commands, create a DataQueue and define what diag packets are to be collected, as well as what fields (such as SummaryText, PacketName, etc.) are to be collected.

QUTS uses a callback mechanism to let the users know whenever there is new data in the queue. Data can be read from the queue as frequently as needed and in as many data packets as needed.

3.4.1 Set filters

Filters can be set in QUTS ahead of time so that QUTS only collects the data specified and saves in a DataQueue. The filters can be based on the PacketType, PacketID, or RegEx on the Summary/Parsed text ,etc.

```
def CreateFilters(allFilters):
    diagPacketFilter = Common.ttypes.DiagPacketFilter()
    diagPacketFilter.idOrNameMask = {}
    for packetType in allFilters:

        listOfID = []
        for id in allFilters[packetType]: ##for string value create a
IdOrName
            listOfID.append(Common.ttypes.DiagIdFilterItem(idOrName=id))
        diagPacketFilter.idOrNameMask[packetType] =   listOfID
    return diagPacketFilter

  allFilters = {}
    ## allFilters [Common.ttypes.DiagPacketType.RESPONSE] = ["125","124"]
    ## allFilters [Common.ttypes.DiagPacketType.SUBSYS_REQUEST] =  ["50/6"]

    ## allFilters [Common.ttypes.DiagPacketType.SUBSYS_RESPONSE] =
[ "75/9483"]
    ## allFilters [Common.ttypes.DiagPacketType.DEBUG_MSG] = ["0005/0002"]

    allFilters [Common.ttypes.DiagPacketType.SUBSYS_RESPONSE] =
["18/2058", "11/37"], "18/93","75/9483"]
    ##item[Common.ttypes.DiagPacketType.EVENT] = ["2803","2804"]
    diagPacketFilter =  CreateFilters(allFilters)
```

Filters can be added or deleted from the queue even after the queue is created. For details on adding/deleting filters, refer to the *Qualcomm Unified Tools Service (QUTS) User Guide* (80-PG522-3).

3.4.2 Set up a DataQueue

Once the filters and what information needs to be saved have been determined, the DataQueue can be setup. There can be multiple DataQueues with each one having its own name, filters, and return config definition.

```
def CreateDataQueueForMonitoring(diagService, allFilters, queueName):
    ### Createa data Queue. Reading will be done in the callback.
    diagPacketFilter =  CreateFilters(allFilters)

    returnObjDiag = Common.ttypes.DiagReturnConfig()
```

```
    returnObjDiag.flags = Common.ttypes.DiagReturnFlags.PARSED_TEXT |
Common.ttypes.DiagReturnFlags.PACKET_NAME |
Common.ttypes.DiagReturnFlags.PACKET_ID |
Common.ttypes.DiagReturnFlags.PACKET_TYPE
|Common.ttypes.DiagReturnFlags.TIME_STAMP_STRING
|Common.ttypes.DiagReturnFlags.RECEIVE_TIME_STRING  |
Common.ttypes.DiagReturnFlags.SUMMARY_TEXT

    ##diagReturnConfig = Common.ttypes.DiagReturnConfig ()
    ##diagReturnConfig.diagConfig = returnObjDiag

    errorCode = diagService.createDataQueue(queueName,
diagPacketFilter,returnObjDiag);
    if(errorCode != 0):
        print ("Error  creating data queue" , errorCode)
    else :
        print ("Data queue Created" )
```

3.4.3 Enable callbacks

QUTS uses a callback mechanism to let users know that there is data in the DataQueue. Only the data that user specified based on the filters set when creating the DataQueue is be placed in the queue.

The script needs to register a callback function to receive notifications. Only one callback function is needed to receive notifications even if there multiple DataQueues. The notification includes the name of the queue that was updated.

```
   ##Set the callback functions to receive notification whenever the
Dataqueue is changed.
    quts_client.setOnDataQueueUpdatedCallback(onDataQueueUpdated)

def onDataQueueUpdated(queueName, queueSize):
    print (queueName , " update " , queueSize)
```

Notice that the callback is set using the quts_client handle and not using the diagService handle. QUTS supports many types of queues and this callback is called when any of those queues is updated. When the callback function is called, the name of the queue is also be included.

3.4.4 Read data from a DataQueue

Once a DataQueue is created, QUTS starts to collect data that matches the criteria specified in the queue definition. Data can be read from the DataQueue whenever the OnQueueUpdated callback is called or at specified intervals set by the user.

The following example command tries to read 2 packets from the queue within the next 5 ms. If 2 packets are not received within the time specified, QUTS times out. If there are more than 2 packets, QUTS returns only 2 and the remaining packets stay in the queue until the queue is cleared.

```
diagPackets = diagService.getDataQueueItems(dataqueueName, 2, 5)
```

The returned diagPackets is a list of diag packets. Each packet contains only the information specified in the returnConfig parameter when the queue is created. All the other information is set to none.

The items in DataQueue can be cleared using the clearDataQueue API.

3.4.5 Delete a DataQueue

Once the purpose for a DataQueue is complete, it is recommended it be deleted using the deleteDataQueue API.

3.5 Save logs

The collected data (all the data that was sent from the device, not just data that was collected using DataQueues), can be saved to a file for postprocessing. The extension of the file is changed from ISF (in QXDM) to .HDF in QUTS.

- Use the StartLogging() command at the point where you want to begin collecting data.
- Use the saveLogFiles(FolderName) command at the point at which to stop data collecting.
  - QUTS takes the folder name and file name as parameters because it saves the data from each protocol (one or more diag, QMI, ADB, etc.) in a separate file.
  - When a session is saved in the user specified folder, the file name includes the details of the protocols and the device from which the data was collected.

3.6 Support for QDSS

In QUTS, QDSS is automatically enabled.

3.7 Clean up

At the end of the script, use diagService.destroyService to close the diag handle and

Commented [KF1]: Nothing specified after "and" – should something go here?

4 Postprocessing data

QUTS saves data in files with the .HDF extension. .IDF files saved from older versions of QXDM can also be opened and processed in QUTS.

4.1 Open a file

QUTS uses the LogSession concept to process data from files. All data processing related APIs are found in the LogSession service. A LogSession is created for a set of files the user wants to process together. This set of files can include simple data from just one device, one protocol, or data collected over several devices in different protocols.

To open a set of files and start a log session:

```
    logSession = quts_client.openLogSession([filePath]) ##Loads a list of
ISF/HDF files. Can be file name or foldername.
```

4.2 Collect data from file(s)

To read the data from the file(s) that are opened in the LogSession, QUTS uses the DataView concept. Users can define filters to select a subset of data. QUTS takes the filters and places only data packets that meet the criteria specified in the DataView. Multiple DataViews can be created, with each having its own filters.

4.2.1 Set filters

4.2.1.1 Filter for protocol

Since a LogSession can have data from multiple devices and protocols, the user must set a filter that specifies what protocol data to view. Filters can include more than one protocol handle (for example, if the device has more than one diag handle, both or either handle can be used). If the user wants to see both diag and QMI data, both diag and QMI handles can be used.

```
    dataPacketFilter = LogSession.ttypes.DataPacketFilter()

   ##Gets the list of devices used in collecting the data in the files in
LogSession
    deviceList = logSession.getDeviceList()
```

```
   ##list of protocols used in the logSession
    listOfProtocols =
logSession.getProtocolList(deviceList[0].deviceHandle)

   ##Get the Diag protocols handle.
    for i in range(len(listOfProtocols)):
         if(listOfProtocols[i].protocolType ==
Common.ttypes.ProtocolType.PROT_DIAG):
              dataPacketFilter.protocolHandleList =
[listOfProtocols[i].protocolHandle]
```

4.2.1.2 Filter for diag data

The following code shows how to set filters for what diag data should be shown. Filters can be set on packet type, ID, or the parsed or summary text.

```
   ##multiple filters
    allFilters = {}

    allFilters [Common.ttypes.DiagPacketType.RESPONSE] = ["115"]
    allFilters [Common.ttypes.DiagPacketType.SUBSYS_RESPONSE] =
[ "75/9483", "18/2048", "33/64005"]
    allFilters [Common.ttypes.DiagPacketType.LOG_PACKET] =
[ "0xB134","0xB11C","0xB13D"]
    allFilters [Common.ttypes.DiagPacketType.DEBUG_MSG] =   [ "32/3"]
    allFilters [Common.ttypes.DiagPacketType.EVENT] =   [ "2693"]
##Set filters for the actual Diag packets.
    diagPacketFilter = Common.ttypes.DiagPacketFilter()
    diagPacketFilter.idOrNameMask = {}
    for packetType in allFilters:

        listOfID = []
        for id in allFilters[packetType]: ##for string value create a
IdOrName
            listOfID.append(Common.ttypes.DiagIdFilterItem(idOrName=id))
        diagPacketFilter.idOrNameMask[packetType] =   listOfID


    dataPacketFilter.diagFilter = diagPacketFilter;
```

4.2.1.3 Filter data using RegEx

In addition to filtering data by packet type, data can be filtered by RegEx as well as field queries:

```
    ##filtering with multiple messages for same packettype. filter by ID or
name
   ## this filter will include items that matches any of the following
filters
    debugFilters = []
    debugFilters.append(Common.ttypes.DiagIdFilterItem(idOrName="LTE
ML1/High"))

debugFilters.append(Common.ttypes.DiagIdFilterItem(idOrName="9504/0002"))
    debugFilters.append(Common.ttypes.DiagIdFilterItem(idOrName="LTE
MACCTRL/High",summaryRegexFilter="lte_mac_qos.c   4808")) ##
    diagPacketFilter.idOrNameMask[Common.ttypes.DiagPacketType.DEBUG_MSG] =
debugFilters
```

In this example, the debug messages are filtered for 3 different scenarios as highlighted.

4.2.2 Set return values

At the time of setting the filter, the user can also set what information from the diag packets they want to receive. This includes packet ID, parsed or summary text, time stamps, etc.

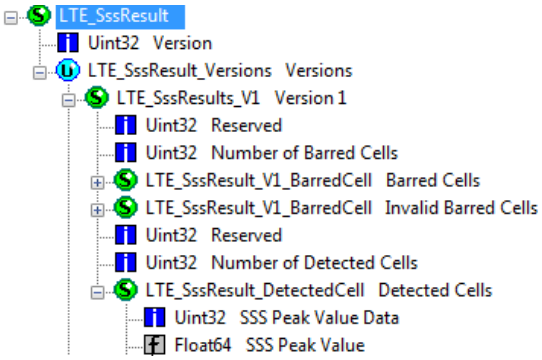
```
   ##Add the fileds you are interested in processing.
    returnObjDiag = Common.ttypes.DiagReturnConfig()

    returnObjDiag.flags = Common.ttypes.DiagReturnFlags.PARSED_TEXT |
Common.ttypes.DiagReturnFlags.PACKET_NAME |
Common.ttypes.DiagReturnFlags.PACKET_ID |
Common.ttypes.DiagReturnFlags.PACKET_TYPE

    packetReturnConfig = LogSession.ttypes.PacketReturnConfig ()
    packetReturnConfig.diagConfig = returnObjDiag
```

4.2.2.1 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

The following sample code defines what fields to show for two different log packets. The columns the user wants to see are highlighted.

```
    ## define what needs to be returned
    returnObjDiag = Common.ttypes.DiagReturnConfig()

   ##Add the fields you are interested in processing.
    returnObjDiag.flags = Common.ttypes.DiagReturnFlags.PARSED_TEXT |
Common.ttypes.DiagReturnFlags.PACKET_NAME |
Common.ttypes.DiagReturnFlags.PACKET_ID |
Common.ttypes.DiagReturnFlags.PACKET_TYPE

    ## For any given id, list the fields you are interested in. Multiple
packets with its own list of fields can be provided.
    diagReturns =  Common.ttypes.DiagReturns()
    diagReturns.flags = returnObjDiag.flags
    ## the fields listed in queries will be displayedin the JSON query
output
```

```
    diagReturns.queries = ['."BSR Trig"', '."LC ID"', '."HARQ ID"',
'."LEN"'] ## ['."Cyclic Shift of DMRS Symbols Slot 0"', '."Cyclic Shift of
DMRS Symbols Slot 1"' ]

    diagReturns1 =  Common.ttypes.DiagReturns()
    diagReturns1.flags = returnObjDiag.flags
    ## the fields listed in queries will be displayedin the JSON query
output
    diagReturns1.queries = ['."Tx Resampler"']
    returnObjDiag.fieldQueries =
{ Common.ttypes.DiagPacketType.LOG_PACKET : {'0xB064' : diagReturns,
'0xB139':diagReturns1} } ##returns two different set of fields based on the
logPacket.
```

4.2.2.2 Sort data by diag time

By default, the data in HDF files is organized by the timestamp received by the PC. As an alternative, data can be sorted by diag time. To enable this function, set the following property:

```
returnObjDiag.diagTimeSorted = True
```

4.2.3 Set up a DataView

Once the filters and return values have been set, the DataViews can be set up. There can be multiple DataViews, with each one having its own name, filters, and return config definition. To set up a data view:

```
    logSession.createDataView("myDataView",
dataPacketFilter,packetReturnConfig);
```

4.2.4 Read data from a view

When a DataView is created, QUTS begins processing the data in the opened files based on the filters provided. The user can read the data while QUTS is continuing to process it.

Data can be read in multiple ways. The following code reads the data over a range of index.

```
    packetRange = LogSession.ttypes.PacketRange()
    packetRange.beginIndex = 0
    packetRange.endIndex =
logSession.getDataPacketCount(dataPacketFilter.protocolHandleList[0])

    dataPackets = logSession.getDataViewItems("data", packetRange)
```

```
    print ("Packets received ", len(dataPackets))
   ##DataPackets has a collection of different types of packets for each
protocol such Diag. QMI etc.
    for i in range(len(dataPackets)):
       if(dataPackets[i].diagPacket is not None):
            print(dataPackets[i].diagPacket.packetId,  '
',dataPackets[i].diagPacket.parsedText )
```

4.2.5 Delete a DataView

Once the purpose for a DataView is complete, it is recommended it be deleted using the deleteDataViewAPI.

4.3 FAQ

4.3.1 Can we use multiple instance of QUTS to process data?

No, QUTS will run as a single QUTS.exe.

4.3.2 What is the common cause of slowness in post processing?

1. User script may be using a filter that is not small enough. Try to exclude log masks from the filter if they are not needed for processing.
2. User script maybe using a return config that enables more than enough. User should try to enable only those fields that are needed for post processing. For example, returning the parsed text for every packet can be expensive.

4.3.3 Is QUTS performance related to CPU or memory?

In general yes for both. QUTS may run multiple-thread tasks that will likely perform better with more CPU cores. For example, getDataViewItems() will distribute processing on as many threads as there are physical CPUs. Higher memory is also recommended for performance, especially when many other applications may be running. Some function calls will consume significant memory. For example, getDataViewItems() could return 10K items per call, and QUTS expect there is enough memory for them.

4.3.4 Will multiThreadedClient boost performance?

The multithreadedClient flag has been shown to have lower performance. Performance wise it is recommended not to turn it on.

4.3.5 How do I get all the packet ID for all the EVENT in a log file?

We would recommend against retrieving all events unless you need to use every one of them in processing. Larger than needed filter would slow down QUTS automation. If all events are really needed, you can specify an empty event filter and QUTS will return all events.

4.3.6 How to merge multiple log file into a single file?

QUTS can open multiple files from the same session. These files can be from different protocols streamed from the same or multiple devices. There is no need to merge them. It is an enhancement in QUTS that logs from different protocols can be separated for better analysis.

4.3.7 Can we convert all fields in packets to JSON format in output file?

First it is expensive to convert packets to either JSON or simple text files. It is recommended to process them in the script on the fly. Secondly, if converting to JSON is needed, it is recommended to convert only the fields you are interested in, instead of all the fields in a packet. You can specify the fields you are interested in so those will be produced in JSON format.

A References

A.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Qualcomm Unified Tools Service (QUTS) API Guide	80-PG522-1
Qualcomm Unified Tools Service (QUTS) User Guide	80-PG522-3

A.2 Acronyms and terms

Acronym or term	Definition
QUTS	Qualcomm Unified Tools Service