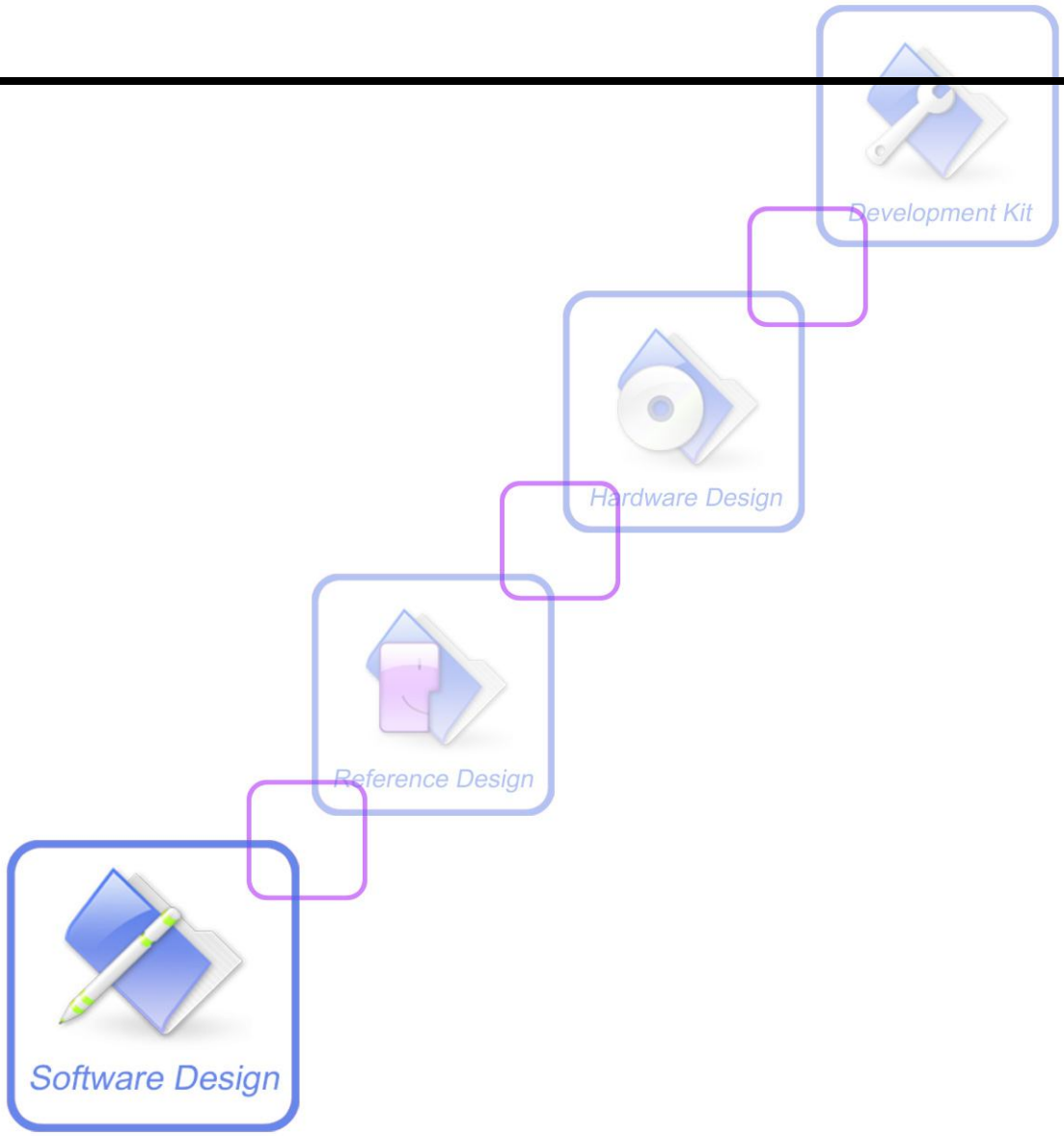
SIM7100 GOBI-API Specification 1.0
（Linux）



Document Title:	SIM7100 GOBI-API Specification
Version:	1.0
Date:	2014-11-21
Status:	Release
Document ID:	SIM7100 GOBI-API Specification（Linux）

Version History

Version	Chapter	Comments
V1.0	New Version	

▪Introducton

1.1. Purpose and Scope

This document covers the GOBI GOBI-API implementation for the Linux. This document is intended for authors of applications that must interface with Qualcomm Gobi-compliant devices.

1.2. References

Document name	Version
80-ND600-2_A_QMI_COMMON_MPSS_DI_1_0.pdf	80-ND600-2 A
GobiConnectionMgmtAPI.pdf	80-VF219-20_C Draft
80-ND600-9_C_QMI_WMS_MPSS_DI_1_0.pdf	80-ND600-9_C
80-ND600-4_B_QMI_DMS_MPSS_DI_1_0.pdf	80-ND600-4_B
80-ND600-5_C_QMI_WDS_MPSS_DI_1_0.pdf	80-ND600-5_C
80-ND600-6_D_QMI_NAS_MPSS_DI_1_0.pdf	80-ND600-6_D
80-ND600-12_B_QMI_UIM_MPSS_DI_1_0.pdf	80-ND600-12_B
80-ND600-15_B_QMI_PBM_MPSS_DI_1_0.pdf	80-ND600-15_B
80-ND600-38_B_QMI_PDC_MPSS_DI_1_0.pdf	80-ND600-38_B

1.3. Glossarys

Abbreviation	Description
GOBI-API	The Qualcomm Gobi™ Connection Management API
QMI	Qualcomm messaging interface
TLV	type-length-value
NAS	Network Access Service
WMS	Wireless Messaging Service
WDS	Wireless Data Service
DMS	Device Management Service
PBM	Phone Book Manager
UIM	User Identity Module
PDC	Persistent Device Configuration

2. GOBI-API functions

GOBI-API functions that utilize Qualcomm's messaging interface are provided in C/C++ headers GobiConnectionMgmtAPI.h, GobiConnectionMgmtStructs.h, GobiConnectionMgmtEnums.h and so file lib GobiConnectionMgmt.so in GOBI directory. There are also some GOBI-API related files under this dire ctory.

Optional parameters

Providing input and output parameters is optional. If you do not care about the response beyond t he error code you do not need to pass in output parameters (i.e., you can pass in 0). If you do not ne ed to specify input, then the input parameters may be 0 as well.

Function timeouts

The library imposes a minimum acceptable timeout of 2 sec and a maximum timeout of 5 min.

Input/output arrays

GOBI-API functions use input and output arrays to manage requests and responses. The arrays are described using QMI message structures. QMI message structures consist of one or more TLV triplets as shown in the following prototype.

TLV prototype

```
// Structure to represent a QMI service Type/Length/Value triplet
struct sQMITLV
{
UINT8 mTypeID;
UINT16 mLength;
// Variable length content of 'mLength' size
// UINT8 mValue[1];
};
```

2.1. GOBI-API function prototype

This function takes an input buffer that should be sufficiently large to hold a request referenced us ing one or more structures defined in GobiConnectionMgmtStructs.h. The response buffer should be suff iciently large to hold a response referenced using one or more structures defined in GobiConnectionMg mtStructs.h..

Prototype

```
ULONG <GOBI-API Function Name>(
GOBIHANDLE handle,
ULONG to,
ULONG inLen,
const BYTE * pIn,
ULONG * pOutLen,
BYTE * pOut )
```

Parameters

Type	Variable	Mode	Description
ULONG	handle	IN	Gobi interface handle
ULONG	to	IN	Timeout for transaction (in milliseconds)

ULONG	inLen	IN	Length of input buffer
const BYTE *	pIn	IN	Input buffer array of TLVs
ULONG *	pOutLen	IN/OUT	Upon input the maximum number of BYTEs pOut can contain, upon output the number of BYTEs copied to pOut
BYTE *	pOut	IN/OUT	Output buffer array of TLVs

2.2. Interface list

MODULE	INTERFACE	Description	ref
COMMON	GobiConnect	This function connects the CM API library to the specified Gobi device	
	GobiCancel	This function cancels the most recent outstanding request for the specified QMI service	
	GobiDisconnect	This function disconnects the CM API library from the currently connected Gobi device	
	SetGenericCallback	This function enables/disables a generic callback	
WMS	WMSSetEventReport	Sets the WMS event reporting conditions for the control point.	
	WMSRawSend	Sends a new message in its raw format	
	WMSRawWrite	Writes a new message given in its raw format.	
	WMSRawRead	Reads a message from the device memory storage and returns the message in its raw format	
	WMSModifyTag	Modifies the metadata tag of a message in the MSM device storage	
	WMSDelete	Deletes the message in a specified memory location	
	WMSListMessages	Requests a list of WMS message indices and meta information within the specified memory storage, matching a specified message tag.	
	WMSGetStorageMaxSize	Queries the maximum number of messages that can be stored per memory	

		storage, as well as the number of slots currently available.	
	WMSGetSMSCAddress	Queries the currently configured SMSC address	
	WMSSetMemoryStatus	Indicates whether the client has storage available for new SMS messages.	
	WMSSetPrimaryClient	Allows the client to set or unset itself as the primary client of QMI_WMS	
PBM	PBMSetIndicationRegistrationState	Sets the registration state for different QMI_PBM indications for the requesting control point	
	PBMGetCapabilities	Returns the capabilities of the PB requested	
	PBMReadRecords	Initiates the Record Read operation by specifying the range of the records to be read	
	PBMWriteRecord	Adds a new record or modifies an existing record	
	PBMDeleteRecord	Deletes a PB record.	
NAS	NASGetHomeNetwork	Retrieves information about the home network of the device	
	NASGetSystemInfo	Provides the system information	
	NASSetRegistrationEventReport	Sets the registration state for different QMI_NAS indications for the requestin g control point	
	NASGetSignalInfo	Queries information regarding the signal strength	
	NASGetPLMNName	Queries the operator name for a specifi ed network	
UIM	UIMVerifyPIN	Verifies the PIN before the card conten t is accessed	
	UIMUnblockPIN	Unblocks a blocked PIN using the PU K code	
	UIMGetCardStatus	Retrieves the current status of the card	
DMS	DMSGetDeviceRevision	Requests the device firmware revision i dentification	
	DMSGetDeviceManfacturer	Requests the device the manufacturer in formation	
	DMSGetDeviceModel	Requests the device model identification	
	DMSGetHardwareRevision	Queries the hardware revision of the de vice	
	DMSGetDeviceSerialNumbers	Requests the serial numbers of the devi	

		ce	
WDS	WDSSetEventReport	Sets the wireless data connection state reporting conditions for the requesting control point	
	WDSStartNetworkInterface	Activates a packet data session (if not already started) on behalf of the reques ting control point	
	WDSStopNetworkInterface	Deactivates a packet data session (unless in use by other control points) on behalf of the requesting control point.	
	WDSGetPacketServiceStatus	Queries the current packet data connect ion status	
	WDSGetDataSessionDuration	Queries the duration of the current call	
	WDSGetDataBearerTechnology	Queries the current data bearer technolo gy	
	WDSGetChannelRates	Queries the current bit rate of the pack et data connection	
PDC	PDCReset	Resets the PDC state variables of the requesting control point.	
	PDCRegisterForIndications	Maintains control point registration for service indications	
	PDCGetSelectedConfig	Gets the active configuration	
	PDCSetSelectedConfig	Sets an available configuration for the device	
	PDCListConfigs	Lists the configurations currently available on the device	
	PDCDeleteConfig	Deletes a configuration from the device	
	PDCLoadConfig	Loads the specified configuration to device memory.	
	PDCActivateConfig	Activates a pending configuration for the component	
	PDCGetConfigInfo	Queries additional information for a configuration.	
	PDCGetConfigLimits	Queries the maximum and current sizes for each configuration memory store	

3. Speciation

3.1. Interface Description

3.1.1. GobiConnect

Description	This function connects the CM API library to the specified Gobi device (UNICODE variant)
Parameter	pInterface [I] - Device interface to connect to pServicesCount [I/O] - Upon input the number of QMI services to connect to, upon output

	the number of QMI services successfully connected to pServices [I/O] - Upon input the array of QMI service IDs to connect to, upon output the array of QMI service IDs successfully connected to pHandle [O] - The returned Gobi interface handle
Return Value	0 is success,others are failed.Please see the error code
Constraint	GobiConnectA connects the CM API library to the specified Gobi device interface (ANSI variant)

Example:

```
#define MAX_SERVICE_NUM 6
ULONG status = 1;
// Connect to WDS, DMS, NAS WMS UIM PBM services .Please see the service define 3.4.1
ULONG svc[MAX_SERVICE_NUM] = { 1,2,3,5,11,12};
ULONG svcCount = MAX_SERVICE_NUM;
GOBIHANDLE handle = 0;
status = GobiConnect (pInterface, &svcCount, &svc[0], &handle);
if (status == 0)
{
    if (svcCount == MAX_SERVICE_NUM)
    {
        mhGobi = handle;
    }
    else
    {
        Disconnect();
        status = 1;
    }
}
```

3.1.2. GobiDisconnect

Description	This function disconnects the CM API library from the currently connected Gobi device
Parameter	handle [I] - Gobi interface handle
Return Value	0 is success,others are failed.Please see the error code
Constraint	

Example:

```
ULONG status = 1;

status = GobiDisconnect ( mhGobi );// mhGobi is handle,please see GobiConnect
```

3.1.3. SetGenericCallback

Description	This function enables/disables a generic callback

Parameter	handle [I] - Gobi interface handle svcID [I] - Service ID to monitor, please 3.4.1 msgID [I] - Message ID to look for, please see _IND message define pCallback [I] - Callback function
Return Value	0 is success,others are failed.Please see the error code
Constraint	

Example:

```
ULONG status = 1;

// Configure the callback with the API
status = SetGenericCallback ( mhGobi, 5, 1, pCallback );
```

3.1.4. WMSSetEventReport

Description	Sets the WMS event reporting conditions for the control point. The control point's event reporting state variables are modified according to the settings specified in the TLVs included in the request message. Specified events are communicated to the registered WMS control point via callback function.

Request Structure	**Optional TLVs**
	// Structure to describe request TLV 0x10 for WMSSetEventReport()
	struct sWMSSetEventReportRequest_NewMTMessageIndicator
	{
	INT8 mReportNewMTMessages;
	};

Indication Structure	// Structure to describe indication TLV 0x10 for WMS EventReport
	struct sWMSEventReportIndication_ReceivedMTMessage
	{
	eQMIWMSStorageTypes mStorageType;
	UINT32 mStorageIndex;
	};
	
	// Structure to describe indication TLV 0x11 for WMS EventReport
	struct sWMSEventReportIndication_TransferRouteMTMessage
	{
	INT8 mACKRequired;

```
    UINT32 mTransactionID;
    eQMIWMSMessageFormats mMessageFormat;
    UINT16 mRawMessageLength;

    // This array must be the size specified by mRawMessageLength
    // UINT8 mRawMessage[1];
};

// Structure to describe indication TLV 0x12 for WMS EventReport
struct sWMSEventReportIndication_MessageMode
{
    eQMIWMSMessageProtocols mMode;
};

// Structure to describe indication TLV 0x13 for WMS EventReport
struct sWMSEventReportIndication_ReceivedETWSMessage
{
    eQMIWMSNotificationType mNotificationType;
    UINT16 mRawMessageLength;

    // This array must be the size specified by mRawMessageLength
    // UINT8 mRawMessage[1];
};

// Structure to describe indication TLV 0x14 for WMS EventReport
struct sWMSEventReportIndication_ReceivedETWSPLMNInfo
{
    UINT16 mMobileCountryCode;
    UINT16 mMobileNetworkCode;
};

// Structure to describe indication TLV 0x15 for WMS EventReport
struct sWMSEventReportIndication_ReceivedSMSCAddress
{
    UINT8 mSMSCAddressLength;

    // This array must be the size specified by mSMSCAddressLength
```

	// char mSMSCAddress[1];
	};
	
	// Structure to describe indication TLV 0x16 for WMS EventReport
	struct sWMSEventReportIndication_SMSOnIMS
	{
	INT8 mMessageReceivedFromIMS;
	};

MSG ID	0x0001
Return Value	QMI_ERR_NONE — No error in the request QMI_ERR_INTERNAL — Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG — Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_MISSING_ARG — A required TLV was not provided
Constraint	We need register callback function to receive the event.

Example:

```
ULONG li = 1024;
BYTE req[1024] = { 0 };

WORD tlvx10Sz = sizeof( sWMSSetEventReportRequest_NewMTMessageIndicator );

sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*) req;
pHeader->mTypeID = 0x10;
pHeader->mLength = tlvx10Sz;

li = sizeof( sQMIRawContentHeader );

sWMSSetEventReportRequest_NewMTMessageIndicator * pTLVx10;
pTLVx10 = (sWMSSetEventReportRequest_NewMTMessageIndicator*)(pOut + li);
memset( pTLVx10, 0, tlvx10Sz );

// Set the value
pTLVx10->mReportNewMTMessages = 1;//0 or 1
li += tlvx10Sz;

status = WMSSetEventReport ( mhGobi, 2000, li, &req[0],0,0 );
SetGenericCallback ( mhGobi, 5, 1, WmsEventReportCallback);


void __cdecl WmsEventReportCallback(
    ULONG                           svcID,
    ULONG                           msgID,
    GOBIHANDLE                      /* handle */,
    ULONG                           outLen,
    const BYTE *                    pOut )
```

```
{
    if (svcID != 5 || msgID != 1)
    {
        return;
    }

    ULONG state = ULONG_MAX;
    ULONG cer = ULONG_MAX;
    eQMIWMSStorageTypes eStroeType;
    ULONG inMsgLen = 1024;
    BYTE aMsg[1024]={0};
    int inMsgOK = 0;


    std::map <UINT8, const sQMIRawContentHeader *> tlvs = GetTLVs( &pOut[0], outLen );
    std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x10 );
    if (pIter != tlvs.end())
    {
        const sQMIRawContentHeader * pTmp = pIter->second;
        if (pTmp->mLength >= sizeof (sWMSEventReportIndication_ReceivedMTMessage))
        {
            pTmp++;
            const sWMSEventReportIndication_ReceivedMTMessage * pState =
                (const sWMSEventReportIndication_ReceivedMTMessage *)pTmp;

                eStroeType = pState->mStorageType;
                inMsgOK = 1;
                if(eQMIWMSStorageTypes_NV == pState->mStorageType)
                {
                        g_pPHInbox[g_ActualLocalInboxSize].store_type = pState->mStorageType;
                        g_pPHInbox[g_ActualLocalInboxSize].index = pState->mStorageIndex;
                }
                else
                {
                        g_pSMSOnPC[g_ActualSIMInboxSize].store_type = pState->mStorageType;
                        g_pSMSOnPC[g_ActualSIMInboxSize].index = pState->mStorageIndex;
                }

        }
    }
    pIter = tlvs.find( 0x11 );
    if (pIter != tlvs.end())
    {
          if(1 == inMsgOK)
          {
                const sQMIRawContentHeader * pTmp = pIter->second;
                if (pTmp->mLength >= sizeof (sWMSEventReportIndication_TransferRouteMTMessage))
                  {
                        pTmp++;
                        const sWMSEventReportIndication_TransferRouteMTMessage * pState =
                                (const sWMSEventReportIndication_TransferRouteMTMessage *)pTmp;

                        BYTE * pTemp = (BYTE *)pState+sizeof(sWMSEventReportIndication_Trans
                          ferRouteMTMessage);
                        inMsgLen = pState->mRawMessageLength;
```

```
                        memcpy(aMsg,pTemp,inMsgLen);
                        if(eQMIWMSStorageTypes_NV == eStroeType)
                        {
                            //wms_ts_decode_CDMA_tl(aMsg,inMsgLen,&g_pPHInbox[g_ActualLocalInboxSize]);
                            wms_ts_decode_GW(aMsg,inMsgLen,&g_pPHInbox[g_ActualLocalInboxSize]);
                            g_pPHInbox[g_ActualLocalInboxSize].type = SMS_MT_UNREAD;
                            g_ActualLocalInboxSize++;
                            g_UnreadLocalSMSCount++;
                            g_nSMSLocalInboxShow++;
                            g_nSMSLocalUnreadShow++;
                        }
                        else
                        {
                            //wms_ts_decode_CDMA_tl(aMsg,inMsgLen,&g_pSMSOnPC[g_ActualSIMInboxSize]);
                            wms_ts_decode_GW(aMsg,inMsgLen,&g_pSMSOnPC[g_ActualSIMInboxSize]);
                            g_pSMSOnPC[g_ActualSIMInboxSize].type = SMS_MT_UNREAD;
                            g_ActualSIMInboxSize++;
                            g_UnreadSIMSMSCount++;
                            g_nSMSSIMInboxShow++;
                            g_nSMSSIMUnreadShow++;
                        }
                    }
                }
            }
            else if(1 == inMsgOK)
            {
                if(eQMIWMSStorageTypes_NV == eStroeType)
                {
                    g_pPHInbox[g_ActualLocalInboxSize].type = SMS_MT_UNREAD;
                    if(0 == mGobiCMDLL.WmsRawReadSmsOne(eStroeType,&g_pPHInbox[g_ActualLocalInboxSize]))
                    {
                        g_ActualLocalInboxSize++;
                            g_UnreadLocalSMSCount++;
                        g_nSMSLocalInboxShow++;
                        g_nSMSLocalUnreadShow++;
                        pTheDialog->PostMessage(WM_NEW_SMS_IN,0, 0);
                    }
                    else
                    {
                        memset(&g_pPHInbox[g_ActualLocalInboxSize],0x00,sizeof(g_pPHInbox[g_ActualLocalInboxSize]));
                    }
                }
                else
                {
                    g_pSMSOnPC[g_ActualSIMInboxSize].type = SMS_MT_UNREAD;
                    if(0 == mGobiCMDLL.WmsRawReadSmsOne(eStroeType,&g_pSMSOnPC[g_ActualSIMInboxSize]))
                    {
                        g_ActualSIMInboxSize++;
```

```
                                    g_UnreadSIMSMSCount++;
                                    g_nSMSSIMInboxShow++;
                                    g_nSMSSIMUnreadShow++;
                                    pTheDialog->PostMessage(WM_NEW_SMS_IN,0,  0);
                        }
                        else
                        {
                                memset(&g_pSMSOnPC[g_ActualSIMInboxSize],0x00,sizeof(g_pSMSOnPC[g
_ActualSIMInboxSize]));
                        }
                }
        }
    }
```

3.1.5. WMSRawSend

Description	Sends a new message in its raw format.

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for WMSRawSend()
	struct sWMSRawSendRequest_MessageData
	{
	eQMIWMSMessageFormats mMessageFormat;
	UINT16 mRawMessageLength;
	
	// This array must be the size specified by mRawMessageLength
	// UINT8 mRawMessage[1];
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for WMSRawSend()
	struct sWMSRawSendRequest_ForceOnDC
	{
	INT8 mForceSendOnDC;
	eQMIWMSCDMAServiceOptions mServiceOption;
	};
	
	// Structure to describe request TLV 0x11 for WMSRawSend()
	struct sWMSRawSendRequest_FollowOnDC
	{
	INT8 mDoNotDisconnectDC;
	};
	
	// Structure to describe request TLV 0x12 for WMSRawSend()

	struct sWMSRawSendRequest_LinkControl
	{
	UINT8 mLinkTimerInSeconds;
	};
	
	// Structure to describe request TLV 0x13 for WMSRawSend()
	struct sWMSRawSendRequest_SMSOnIMS
	{
	INT8 mMessageToBeSentOnIMS;
	};
	
	// Structure to describe request TLV 0x14 for WMSRawSend()
	struct sWMSRawSendRequest_RetryMessage
	{
	INT8 mMessageIsARetry;
	};
	
	// Structure to describe request TLV 0x15 for WMSRawSend()
	struct sWMSRawSendRequest_RetryMessageID
	{
	UINT32 mMessageRetryID;
	};

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WMSRawSend()
	struct sWMSRawSendResponse_MessageID
	{
	UINT16 mMessageID;
	};
	Optional TLVs
	// Structure to describe response TLV 0x10 for WMSRawSend()
	struct sWMSRawSendResponse_CauseCode
	{
	eQMIWMSCauseCodes mCauseCode;
	};

	// Structure to describe response TLV 0x11 for WMSRawSend()
	struct sWMSRawSendResponse_ErrorClass
	{
	eQMIWMSErrorClasses mErrorClass;
	};
	
	// Structure to describe response TLV 0x12 for WMSRawSend()
	struct sWMSRawSendResponse_CauseInfo
	{
	eQMIWMSRPCauseCodes mGSMWCDMARPCause;
	eQMIWMSTPCauseCodes mGSMWCDMATPCause;
	};
	
	// Structure to describe response TLV 0x13 for WMSRawSend()
	struct sWMSRawSendResponse_MessageDeliveryFailureType
	{
	eQMIWMSMessageDeliveryFailureType mMessageDeliveryFailureType;
	};
	
	// Structure to describe response TLV 0x14 for WMSRawSend()
	struct sWMSRawSendResponse_MessageDeliveryFailureCause
	{
	eQMIWMSDeliveryFailures mDeliveryFailureCause;
	};
	
	// Structure to describe response TLV 0x15 for WMSRawSend()
	struct sWMSRawSendResponse_CallControlModifiedInfo
	{
	UINT8 mAlphaIDLength;
	
	// This array must be the size specified by mAlphaIDLength
	// UINT8 mAlphaID[1];
	};

Return Value	QMI_ERR_NONE — No error in the request QMI_ERR_INTERNAL — Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG — Message was not formulated correctly by the control point or the message was corrupted during

	transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_ARG_TOO_LONG Argument passed in a TLV was larger than the available storage in the device QMI_ERR_MISSING_ARG A required TLV was not provided QMI_ERR_INVALID_ARG One of the parameters specified contains an invalid value QMI_ERR_CAUSE_CODE SMS cause code QMI_ERR_ENCODING Message is not encoded properly QMI_ERR_INVALID_MESSAGE_ID Message ID specified for the message is invalid QMI_ERR_MESSAGE_NOT_SENT Message could not be sent QMI_ERR_MESSAGE_DELIVERY_FAILURE Message could not be delivered QMI_ERR_DEVICE_NOT_READY Device is not ready to send the message QMI_ERR_NETWORK_NOT_READY Network is not ready to send the message QMI_ERR_OP_DEVICE_UNSUPPORTED Selected operation is not supported by the device QMI_ERR_OP_NETWORK_UNSUPPORTED Selected operation is not supported by the network QMI_ERR_SMSC_ADDR SMSC address specified is invalid QMI_ERR_CALL_FAILED Cannot bring up the CDMA dedicated channel QMI_ERR_MSG_BLOCKED Message is blocked because the recipient is not on the FDN QMI_ERR_INVALID_OPERATION SMS on IMS TLV is set to TRUE; however, IMS is not registered
Constraint	

Example:

```
ULONG status = 1;
ULONG li = 1024;
BYTE req[1024] = { 0 };

li = sizeof( sWMSRawSendRequest_MessageData )
                + (WORD)messageSize;


sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)( req);
pHeader->mTypeID = 0x01;
pHeader->mLength = li;

ULONG offset = sizeof( sQMIRawContentHeader );
```

```
    // The index
    sWMSRawSendRequest_MessageData * pTLVx01;
    pTLVx01 = (sWMSRawSendRequest_MessageData*)( req + li);
    memset( pTLVx01, 0, li );

    // Set the values
    pTLVx01->mMessageFormat = (eQMIWMSMessageFormats)messageFormat;
    pTLVx01->mRawMessageLength = (UINT16)messageSize;

    li += sizeof( sWMSRawSendRequest_MessageData );

    // Add the message
    memcpy( (req + li), pMessage, messageSize );

    li += messageSize;
WMSRawSend ( mhGobi, 2000, li, &req[0],0,0 );
```

3.1.6. WMSRawWrite

Description	Writes a new message given in its raw format.

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for WMSRawWrite()
	struct sWMSRawWriteRequest_MessageData
	{
	eQMIWMSStorageTypes mStorageType;
	eQMIWMSMessageFormats mMessageFormat;
	UINT16 mRawMessageLength;
	
	// This array must be the size specified by mRawMessageLength
	// UINT8 mRawMessage[1];
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for WMSRawWrite()
	struct sWMSRawWriteRequest_MessageTag
	{
	eQMIWMSMessageTags mMessageTag;
	};

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WMSRawWrite()
	struct sWMSRawWriteResponse_MessageIndex
	{
	UINT32 mStorageIndex;

	};
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocated memory to formulate a response QMI_ERR_ARG_TOO_LONG Argument passed in a TLV was larger than the available storage in the device QMI_ERR_MISSING_ARG A required TLV was not provided QMI_ERR_INVALID_ARG One of the parameters contains an invalid value QMI_ERR_ENCODING Message is not encoded properly QMI_ERR_DEVICE_STORAGE_FULL Memory storage specified in the request is full QMI_ERR_DEVICE_NOT_READY Device is not ready to send the message QMI_ERR_OP_DEVICE_UNSUPPORTED Selected operation is not supported by the device QMI_ERR_SMSC_ADDR SMSC address specified is invalid
Constraint	

Example:

```
    ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };

// Check size
    WORD tlvx01Sz = sizeof( sWMSRawWriteRequest_MessageData )
                    + (WORD)messageSize;

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)( req);
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    // The index
    sWMSRawWriteRequest_MessageData * pTLVx01;
    pTLVx01 = (sWMSRawWriteRequest_MessageData*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the values
    pTLVx01->mMessageFormat = (eQMIWMSMessageFormats)messageFormat;
    pTLVx01->mStorageType = (eQMIWMSStorageTypes)inStoreType;
    pTLVx01->mRawMessageLength = messageSize;

    offset += sizeof( sWMSRawWriteRequest_MessageData );
```

```
    // Add the message
    memcpy( (pOut + offset), pMessage, messageSize );

    offset += messageSize;

    // Check size
WORD tlvx10Sz = sizeof( sWMSRawWriteRequest_MessageTag );

pHeader = (sQMIRawContentHeader*)(pOut + offset);
pHeader->mTypeID = 0x10;
pHeader->mLength = tlvx10Sz;

offset += sizeof( sQMIRawContentHeader );

// The SPC
sWMSRawWriteRequest_MessageTag * pTLVx10;
pTLVx10 = (sWMSRawWriteRequest_MessageTag*)(pOut + offset);
memset( pTLVx10, 0, tlvx10Sz );

// Set the values
pTLVx10->mMessageTag = (eQMIWMSMessageTags)inMsgTag;

offset += tlvx10Sz;

li = offset;


WMSRawWrite ( mhGobi, 2000, li, &req[0],&lo,&rsp[0] );

// Find the TLV
    const sWMSRawWriteResponse_MessageIndex * pTLVx01;
    ULONG outLenx01;
    ULONG rc = GetTLV( lo, rsp, 0x01, &outLenx01, (const BYTE **)&pTLVx01 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
    }

    // Is the TLV large enough?
    if (outLenx01 < sizeof( sWMSRawWriteResponse_MessageIndex ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    *pStoreIndex = pTLVx01->mStorageIndex;
```

3.1.7. WMSRawRead

Description	Reads a message from the device memory storage and returns the message in its raw format
Request Structure	**Mandatory TLVs** // Structure to describe request TLV 0x01 for WMSRawRead()

	struct sWMSRawReadRequest_MessageIndex
	{
	eQMIWMSStorageTypes mStorageType;
	UINT32 mStorageIndex;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for WMSRawRead()
	struct sWMSRawReadRequest_MessageMode
	{
	eQMIWMSMessageProtocols mMode;
	};
	
	// Structure to describe request TLV 0x11 for WMSRawRead()
	struct sWMSRawReadRequest_SMSOnIMS
	{
	INT8 mMessageToBeReadFromIMS;
	};

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WMSRawRead()
	struct sWMSRawReadResponse_MessageData
	{
	eQMIWMSMessageTags mMessageTag;
	eQMIWMSMessageFormats mMessageFormat;
	UINT16 mRawMessageLength;
	
	// This array must be the size specified by mRawMessageLength
	// UINT8 mRawMessage[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_MISSING_ARG A required TLV was not provided QMI_ERR_OP_DEVICE_UNSUPPORTED Selected operation is not supported by the device QMI_ERR_INVALID_ARG One of the parameters specified contains an invalid value

	QMI_ERR_INVALID_INDEX	Memory storage index specified in the request is invalid
	QMI_ERR_NO_ENTRY	No message exists at the specified memory storage designation
	QMI_ERR_TDPU_TYPE	Message in memory contains a TPDU type that cannot be read as a raw message
Constraint		

Example:

```
ULONG PackGetSMS(
    ULONG *             pOutLen,
    BYTE *              pOut,
    ULONG               storageType,
    ULONG               messageIndex )
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sWMSRawReadRequest_MessageIndex );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)(pOut);
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    // The index
    sWMSRawReadRequest_MessageIndex * pTLVx01;
    pTLVx01 = (sWMSRawReadRequest_MessageIndex*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the values
    pTLVx01->mStorageType = (eQMIWMSStorageTypes)storageType;
    pTLVx01->mStorageIndex = messageIndex;

    offset += tlvx01Sz;

    //added by limingjun begin
    // Check size
        WORD tlvx10Sz = sizeof( sWMSRawReadRequest_MessageMode );
        if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx10Sz)
        {
            return eGOBI_ERR_BUFFER_SZ;
        }
```

```
        pHeader = (sQMIRawContentHeader*)(pOut + offset);
        pHeader->mTypeID = 0x10;
        pHeader->mLength = tlvx10Sz;

        offset += sizeof( sQMIRawContentHeader );

        // The SPC
        sWMSRawReadRequest_MessageMode * pTLVx10;
        pTLVx10 = (sWMSRawReadRequest_MessageMode*)(pOut + offset);
        memset( pTLVx10, 0, tlvx10Sz );

        // Set the values
        pTLVx10->mMode = (eQMIWMSMessageProtocols)GetCurrentProtocal();

        offset += tlvx10Sz;
    //added by limingjun end
    *pOutLen = offset;

    return eGOBI_ERR_NONE;
}

ULONG ParseGetSMS(
    ULONG                 inLen,
    const BYTE *          pIn,
    ULONG *               pMessageTag,
    ULONG *               pMessageFormat,
    ULONG *               pMessageSize,
    BYTE *                pMessage )
{
    // Validate arguments
    if (pIn == 0
    ||  pMessageTag == 0
    ||  pMessageFormat == 0
    ||  pMessageSize == 0
    ||  *pMessageSize == 0
    ||  pMessage == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    ULONG maxMessageSz = *pMessageSize;

    // Assume failure
    *pMessageSize = 0;

    // Find the messages
    const sWMSRawReadResponse_MessageData * pTLVx01;
    ULONG outLenx01;
    ULONG rc = GetTLV( inLen, pIn, 0x01, &outLenx01, (const BYTE **)&pTLVx01 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
    }
```

```
    if (outLenx01 < sizeof( sWMSRawReadResponse_MessageData ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    *pMessageTag = pTLVx01->mMessageTag;
    *pMessageFormat = pTLVx01->mMessageFormat;

    ULONG messageSz = pTLVx01->mRawMessageLength;
    if (messageSz == 0)
    {
        // No stored messages, but not necessarily a failure
        return eGOBI_ERR_NONE;
    }

    if (messageSz > maxMessageSz)
    {
        messageSz = maxMessageSz;
    }

    // Verify there is room for the array in the TLV
    if (outLenx01 < sizeof( sWMSRawReadResponse_MessageData ) + messageSz)
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }
    BYTE *                pTemp = (BYTE *)pTLVx01;
    memcpy( pMessage,
            pTemp + sizeof( sWMSRawReadResponse_MessageData ),
            messageSz );

    *pMessageSize = messageSz;
    return eGOBI_ERR_NONE;
}


ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };

PackGetSMS(&li,req, eQMIWMSStorageTypes_NV,0);
    status = WMSRawRead( mhGobi, 2000, li, &req[0], &lo, &rsp[0] );
    if (status != 0)
    {
        return status;
    }

    ULONG inMsgLen = 1024;
    BYTE aMsg[1024]={0};
    status = ParseGetSMS( lo,&rsp[0],&uMsgType,&uMsgFormat,&inMsgLen,aMsg);
```

3.1.8. WMSModifyTag

Description	Modifies the metadata tag of a message in the MSM device storage

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for WMSModifyTag()
	struct sWMSModifyTagRequest_MessageTag
	{
	eQMIWMSStorageTypes mStorageType;
	UINT32 mStorageIndex;
	eQMIWMSMessageTags mMessageTag;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for WMSModifyTag()
	struct sWMSModifyTagRequest_MessageMode
	{
	eQMIWMSMessageProtocols mMode;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_INVALID_ARG One of the parameters specified contains an invalid value QMI_ERR_INVALID_INDEX Memory storage index specified in the request is invalid QMI_ERR_NO_ENTRY No message exists at the specified memory storage designation QMI_ERR_MISSING_ARG A required TLV was not provided QMI_ERR_OP_DEVICE_UNSUPPORTED Selected operation is not supported by the device
Constraint	

Example:

```
ULONG  PackModifyTag(
    ULONG  *                    pOutLen,
    BYTE  *                     pOut,
        ULONG  uStorageType,ULONG  uIndex,ULONG  uMsgTag,ULONG  uMod)
```

```
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sWMSModifyTagRequest_MessageTag );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sWMSModifyTagRequest_MessageTag * pTLVx01;
    pTLVx01 = (sWMSModifyTagRequest_MessageTag*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the value
    pTLVx01->mStorageType = (eQMIWMSStorageTypes)uStorageType;
    pTLVx01->mStorageIndex = uIndex;
    pTLVx01->mMessageTag = (eQMIWMSMessageTags)uMsgTag;
    offset += tlvx01Sz;

    // Check size
     WORD tlvx10Sz = sizeof( sWMSModifyTagRequest_MessageMode );
     if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx10Sz)
     {
          return eGOBI_ERR_BUFFER_SZ;
     }

     pHeader = (sQMIRawContentHeader*)(pOut + offset);
     pHeader->mTypeID = 0x10;
     pHeader->mLength = tlvx10Sz;

     offset += sizeof( sQMIRawContentHeader );

     // The SPC
     sWMSModifyTagRequest_MessageMode * pTLVx10;
     pTLVx10 = (sWMSModifyTagRequest_MessageMode*)(pOut + offset);
     memset( pTLVx10, 0, tlvx10Sz );

     // Set the values
     pTLVx10->mMode = (eQMIWMSMessageProtocols)uMod;

     offset += tlvx10Sz;

    *pOutLen = offset;
    return eGOBI_ERR_NONE;
```

```
}

ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };


    PackModifyTag(&li,req, eQMIWMSStorageTypes_NV,1, eQMIWMSMessageTags_MTRead, eQMIWMS
MessageProtocols_GSMWCDMAUnsupported);
    status = WMSModifyTag ( mhGobi, 2000, li, &req[0],&lo,&rsp[0] );
```

3.1.9. WMSDelete

Description	Deletes the message in a specified memory location

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for WMSDelete()
	struct sWMSDeleteRequest_MemoryStorage
	{
	eQMIWMSStorageTypes mStorageType;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for WMSDelete()
	struct sWMSDeleteRequest_MessageIndex
	{
	UINT32 mStorageIndex;
	};
	
	// Structure to describe request TLV 0x11 for WMSDelete()
	struct sWMSDeleteRequest_MessageTag
	{
	eQMIWMSMessageTags mMessageTag;
	};
	
	// Structure to describe request TLV 0x12 for WMSDelete()
	struct sWMSDeleteRequest_MessageMode
	{
	eQMIWMSMessageProtocols mMode;
	};

Return Value	QMI_ERR_NONE No error in the request

	QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_INVALID_ARG One of the parameters specified contains an invalid value QMI_ERR_INVALID_INDEX Memory storage index specified in the request is invalid QMI_ERR_NO_ENTRY No message exists at the specified memory storage designation QMI_ERR_MISSING_ARG A required TLV was not provided QMI_ERR_OP_DEVICE_UNSUPPORTED Selected operation is not supported by the device
Constraint	There are three ways to use this message: • Specify the memory storage only – Deletes all messages from the memory storage • Specify the memory storage and a message tag – Deletes all messages from the memory storage that match the specific message tag • Specify the memory storage and a message index – Deletes only the message at the specific index from the memory storage

Example:

```
ULONG PackDeleteMsg(
    ULONG *                         pOutLen,
    BYTE *                          pOut,
        ULONG uStorageType,ULONG uIndex,ULONG uMsgTag,ULONG uMod)
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sWMSDeleteRequest_MemoryStorage );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
```

```
pHeader->mLength = tlvx01Sz;

ULONG offset = sizeof( sQMIRawContentHeader );

sWMSDeleteRequest_MemoryStorage * pTLVx01;
pTLVx01 = (sWMSDeleteRequest_MemoryStorage*)(pOut + offset);
memset( pTLVx01, 0, tlvx01Sz );

// Set the value
pTLVx01->mStorageType = (eQMIWMSStorageTypes)uStorageType;
offset += tlvx01Sz;

if(uIndex != 0xffffffff)
{
        // Check size
            WORD tlvx10Sz = sizeof( sWMSDeleteRequest_MessageIndex );
            if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx10Sz)
            {
                    return eGOBI_ERR_BUFFER_SZ;
            }

            pHeader = (sQMIRawContentHeader*)(pOut + offset);
            pHeader->mTypeID = 0x10;
            pHeader->mLength = tlvx10Sz;

            offset += sizeof( sQMIRawContentHeader );

            // The SPC
            sWMSDeleteRequest_MessageIndex * pTLVx10;
            pTLVx10 = (sWMSDeleteRequest_MessageIndex*)(pOut + offset);
            memset( pTLVx10, 0, tlvx10Sz );

            // Set the values
            pTLVx10->mStorageIndex = uIndex;

            offset += tlvx10Sz;
}

if(uMsgTag != 0xffffffff)
{
        // Check size
            WORD tlvx11Sz = sizeof( sWMSDeleteRequest_MessageTag );
            if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx11Sz)
            {
                    return eGOBI_ERR_BUFFER_SZ;
            }

            pHeader = (sQMIRawContentHeader*)(pOut + offset);
            pHeader->mTypeID = 0x11;
            pHeader->mLength = tlvx11Sz;

            offset += sizeof( sQMIRawContentHeader );

            // The SPC
            sWMSDeleteRequest_MessageTag * pTLVx11;
```

```
                pTLVx11 = (sWMSDeleteRequest_MessageTag*)(pOut + offset);
                memset( pTLVx11, 0, tlvx11Sz );

                // Set the values
                pTLVx11->mMessageTag = (eQMIWMSMessageTags)uMsgTag;

                offset += tlvx11Sz;
    }


        // Check size
    WORD tlvx12Sz = sizeof( sWMSDeleteRequest_MessageMode );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx12Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x12;
    pHeader->mLength = tlvx12Sz;

    offset += sizeof( sQMIRawContentHeader );

    // The SPC
    sWMSDeleteRequest_MessageMode * pTLVx12;
    pTLVx12 = (sWMSDeleteRequest_MessageMode*)(pOut + offset);
    memset( pTLVx12, 0, tlvx12Sz );

    // Set the values
    pTLVx12->mMode = (eQMIWMSMessageProtocols)uMod;

    offset += tlvx12Sz;

   *pOutLen = offset;
    return eGOBI_ERR_NONE;
}

ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };


    PackDeleteMsg(&li,req,uStorageType,uIndex,uMsgTag,uMod);
    status = mpFnWmsDeleteMsg( mhGobi, 2000, li, &req[0],&lo,&rsp[0] );
```

3.1.10. WMSListMessages

Description	Requests a list of WMS message indices and meta information within the specified memory storage, matching a specified message tag.
Request Structure	**Mandatory TLVs** // Structure to describe request TLV 0x01 for WMSListMessages()

	struct sWMSListMessagesRequest_MemoryStorage
	{
	eQMIWMSStorageTypes mStorageType;
	};
	
	Optional TLVs
	// Structure to describe request TLV 0x10 for WMSListMessages()
	struct sWMSListMessagesRequest_MessageTag
	{
	eQMIWMSMessageTags mMessageTag;
	};
	
	// Structure to describe request TLV 0x11 for WMSListMessages()
	struct sWMSListMessagesRequest_MessageMode
	{
	eQMIWMSMessageProtocols mMode;
	};

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WMSListMessages()
	struct sWMSListMessagesResponse_MessageList
	{
	UINT32 mNumberOfMessages;
	
	struct sMessage
	{
	UINT32 mStorageIndex;
	eQMIWMSMessageTags mMessageTag;
	};
	
	// This array must be the size specified by mNumberOfMessages
	// sMessage mMessages[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission

	QMI_ERR_MISSING_ARG	A required TLV was not provided
	QMI_ERR_INVALID_ARG	One of the parameters specified contains an invalid value
	QMI_ERR_OP_DEVICE_UNSUPPORTED	Selected operation is not supported by the device
Constraint		

Example:

```
ULONG PackGetSMSList(
    ULONG *              pOutLen,
    BYTE *               pOut,
    ULONG                storageType,
    ULONG *              pRequestedTag )
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sWMSListMessagesRequest_MemoryStorage );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)(pOut);
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    // The storage type
    sWMSListMessagesRequest_MemoryStorage * pTLVx01;
    pTLVx01 = (sWMSListMessagesRequest_MemoryStorage*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the values
    pTLVx01->mStorageType = (eQMIWMSStorageTypes)storageType;

    offset += tlvx01Sz;

    // Add the Message tag, if specified
    if (pRequestedTag != 0)
    {
        // Check size
        WORD tlvx10Sz = sizeof( sWMSListMessagesRequest_MessageTag );
        if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx10Sz)
        {
            return eGOBI_ERR_BUFFER_SZ;
        }

        pHeader = (sQMIRawContentHeader*)(pOut + offset);
```

```
        pHeader->mTypeID = 0x10;
        pHeader->mLength = tlvx10Sz;

        offset += sizeof( sQMIRawContentHeader );

        // The SPC
        sWMSListMessagesRequest_MessageTag * pTLVx10;
        pTLVx10 = (sWMSListMessagesRequest_MessageTag*)(pOut + offset);
        memset( pTLVx10, 0, tlvx10Sz );

        // Set the values
        pTLVx10->mMessageTag = (eQMIWMSMessageTags)*pRequestedTag;

        offset += tlvx10Sz;
    }

    //added by limingjun
    // Check size
        WORD tlvx11Sz = sizeof( sWMSListMessagesRequest_MessageMode );
        if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx11Sz)
        {
            return eGOBI_ERR_BUFFER_SZ;
        }

        pHeader = (sQMIRawContentHeader*)(pOut + offset);
        pHeader->mTypeID = 0x11;
        pHeader->mLength = tlvx11Sz;

        offset += sizeof( sQMIRawContentHeader );

        // The SPC
        sWMSListMessagesRequest_MessageMode * pTLVx11;
        pTLVx11 = (sWMSListMessagesRequest_MessageMode*)(pOut + offset);
        memset( pTLVx11, 0, tlvx11Sz );

        // Set the values
        pTLVx11->mMode = (eQMIWMSMessageProtocols)GetCurrentProtocal();

        offset += tlvx11Sz;

    *pOutLen = offset;
    return eGOBI_ERR_NONE;
}

ULONG ParseGetSMSList(
    ULONG                 inLen,
    const BYTE *          pIn,
    ULONG storageType)
{
    // Validate arguments
    if (pIn == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }
```

```
// Find the messages
const sWMSListMessagesResponse_MessageList * pTLVx01;
ULONG outLenx01;
ULONG rc = GetTLV( inLen, pIn, 0x01, &outLenx01, (const BYTE **)&pTLVx01 );
if (rc != eGOBI_ERR_NONE)
{
    return rc;
}

if (outLenx01 < sizeof( sWMSListMessagesResponse_MessageList ))
{
    return eGOBI_ERR_MALFORMED_RSP;
}

ULONG messageListSz = pTLVx01->mNumberOfMessages;
if (messageListSz == 0)
{
    // No stored messages, but not necessarily a failure
    return eGOBI_ERR_NONE;
}


const sWMSListMessagesResponse_MessageList::sMessage * pMessages;

// Verify there is room for the array in the TLV
if (outLenx01 < sizeof( sWMSListMessagesResponse_MessageList )
                + sizeof( sWMSListMessagesResponse_MessageList::sMessage )
                  * messageListSz)
{
    return eGOBI_ERR_MALFORMED_RSP;
}

// Align to the first array element
pMessages = (const sWMSListMessagesResponse_MessageList::sMessage *)
            ((const BYTE *)pTLVx01
            + sizeof( sWMSListMessagesResponse_MessageList ));

if(eQMIWMSStorageTypes_UIM == storageType)
{
        for (ULONG m = 0; m < messageListSz; m++)
        {
                g_ActualSIMSMSAllSize++;
                switch(pMessages->mMessageTag)
                {
                    case eQMIWMSMessageTags_MTRead:
                        g_pSMSOnPC[g_ActualSIMInboxSize].type = SMS_MT_READ;
                        g_pSMSOnPC[g_ActualSIMInboxSize].index = pMessages->mStorageIndex;
                        g_pSMSOnPC[g_ActualSIMInboxSize].store_type = SMS_LOCATION_SIM;
                        g_ActualSIMInboxSize++;
                            g_nSMSSIMInboxShow++;
                        break;

                     case eQMIWMSMessageTags_MTNotRead:
                        g_pSMSOnPC[g_ActualSIMInboxSize].type = SMS_MT_UNREAD;
                        g_pSMSOnPC[g_ActualSIMInboxSize].index = pMessages->mStorageIndex;
```

```
                        g_pSMSOnPC[g_ActualSIMInboxSize].store_type = SMS_LOCATION_SIM;
                        g_ActualSIMInboxSize++;
                            g_UnreadSIMSMSCount++;
                            g_nSMSSIMInboxShow++;
                            g_nSMSSIMUnreadShow++;
                        break;

                    case eQMIWMSMessageTags_MOSend:
                      g_SentSMSArray[g_ActualSIMOutboxSize].type = SMS_MO_SENT;
                      g_SentSMSArray[g_ActualSIMOutboxSize].index = pMessages->mStorageIndex;
                      g_SentSMSArray[g_ActualSIMOutboxSize].store_type = SMS_LOCATION_SIM;
                      g_ActualSIMOutboxSize++;
                      g_nSMSSIMOutboxShow++;
                     break;

                      case eQMIWMSMessageTags_MONotSent:
                      g_SentSMSArray[g_ActualSIMOutboxSize].type = SMS_MO_UNSENT;
                      g_SentSMSArray[g_ActualSIMOutboxSize].index = pMessages->mStorageIndex;
                      g_SentSMSArray[g_ActualSIMOutboxSize].store_type = SMS_LOCATION_SIM;
                      g_ActualSIMOutboxSize++;
                      g_nSMSSIMOutboxShow++;
                     break;
                }
               pMessages++;
        }
}
else
{
        for (ULONG m = 0; m < messageListSz; m++)
        {
                g_ActualLocalSMSAllSize++;
                switch(pMessages->mMessageTag)
                {
                case eQMIWMSMessageTags_MTRead:
                        g_pPHInbox[g_ActualLocalInboxSize].type = SMS_MT_READ;
                        g_pPHInbox[g_ActualLocalInboxSize].index = pMessages->mStorageIndex;
                        g_pPHInbox[g_ActualLocalInboxSize].store_type = SMS_LOCATION_LOCAL;
                        g_ActualLocalInboxSize++;
                        g_nSMSLocalInboxShow++;
                        break;

                 case eQMIWMSMessageTags_MTNotRead:
                        g_pPHInbox[g_ActualLocalInboxSize].type = SMS_MT_UNREAD;
                        g_pPHInbox[g_ActualLocalInboxSize].index = pMessages->mStorageIndex;
                        g_pPHInbox[g_ActualLocalInboxSize].store_type = SMS_LOCATION_LOCAL;
                        g_ActualLocalInboxSize++;
                        g_UnreadLocalSMSCount++;
                        g_nSMSLocalInboxShow++;
                        g_nSMSLocalUnreadShow++;
                        break;

                  case eQMIWMSMessageTags_MOSend:
                      g_pPHOutbox[g_ActualLocalOutboxSize].type = SMS_MO_SENT;
                      g_pPHOutbox[g_ActualLocalOutboxSize].index = pMessages->mStorageIndex;
                      g_pPHOutbox[g_ActualLocalOutboxSize].store_type = SMS_LOCATION_LOCAL;
                      g_ActualLocalOutboxSize++;
```

```
                        g_nSMSLocalOutboxShow++;
                        break;

                    case eQMIWMSMessageTags_MONotSent:
                        g_pPHOutbox[g_ActualLocalOutboxSize].type = SMS_MO_UNSENT;
                        g_pPHOutbox[g_ActualLocalOutboxSize].index = pMessages->mStorageIndex;
                          g_pPHOutbox[g_ActualLocalOutboxSize].store_type = SMS_LOCATION_LOCAL;
                        g_ActualLocalOutboxSize++;
                        g_nSMSLocalOutboxShow++;
                        break;
                }
                pMessages++;
            }
    }


    return eGOBI_ERR_NONE;
}

 ULONG li = 1024;
     BYTE req[1024] = { 0 };
     ULONG lo = 8096;
     BYTE rsp[8096] = { 0 };


        for(ULONG eMsgTag = eQMIWMSMessageTags_MTRead; eMsgTag<=eQMIWMSMessageTags_
MONotSent; eMsgTag++)
        {
                li = 1024;
                lo = 8096;
                memset(req,0x00,sizeof(req));
                memset(rsp,0x00,sizeof(rsp));

                PackGetSMSList(&li,req,storageType,&eMsgTag);
                status = WMSListMessages ( mhGobi, 2000, li, &req[0], &lo, &rsp[0] );
                if (status != 0)
                {
                        return status;
                }

                //ULONG uMessageListSize = g_VolumnSMSInSimAll;
                /*BYTE *pMessageList = NULL;

                if((eQMIWMSMessageTags_MTRead == eMsgTag)||(eQMIWMSMessageTags_MTNotRea
d == eMsgTag))
                {
                        pMessageList = (BYTE*)g_pSMSOnPC;
                }
                else
                {
                        pMessageList = (BYTE*)g_SentSMSArray;
                }*/
                status = ParseGetSMSList( lo,&rsp[0],storageType);
        }
```

3.1.11. WMSGetStorageMaxSize

Description	Queries the maximum number of messages that can be stored per memory storage, as well as the number of slots currently available.

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for WMSGetStorageMaxSize()
	struct sWMSGetStorageMaxSizeRequest_MemoryStorage
	{
	eQMIWMSStorageTypes mStorageType;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for WMSGetStorageMaxSize()
	struct sWMSGetStorageMaxSizeRequest_MessageMode
	{
	eQMIWMSMessageProtocols mMode;
	};

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WMSGetStorageMaxSize()
	struct sWMSGetStorageMaxSizeResponse_MaxSize
	{
	UINT32 mMaxStorageSizeInMessages;
	};
	Optional TLVs
	// Structure to describe response TLV 0x10 for WMSGetStorageMaxSize()
	struct sWMSGetStorageMaxSizeResponse_AvailableSize
	{
	UINT32 mFreeStorageSizeInMessages;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_MISSING_ARG A required TLV was not provided QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_OP_DEVICE_UNSUPPORTED Selected operation is not supported by the device

	QMI_ERR_INVALID_ARG One of the parameters specified contains an invalid value
Constraint	

Example:

```
ULONG PackGetSmsMaxSize(
  ULONG *                       pOutLen,
   BYTE *                       pOut,
   ULONG          uStorageType ,
   ULONG          uMode )
{
   // Validate arguments
   if (pOut == 0)
   {
      return eGOBI_ERR_INVALID_ARG;
   }

   // Check size
   WORD tlvx01Sz = sizeof( sWMSGetStorageMaxSizeRequest_MemoryStorage );
   if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
   {
      return eGOBI_ERR_BUFFER_SZ;
   }

   sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
   pHeader->mTypeID = 0x01;
   pHeader->mLength = tlvx01Sz;

   ULONG offset = sizeof( sQMIRawContentHeader );

   sWMSGetStorageMaxSizeRequest_MemoryStorage * pTLVx01;
   pTLVx01 = (sWMSGetStorageMaxSizeRequest_MemoryStorage*)(pOut + offset);
   memset( pTLVx01, 0, tlvx01Sz );

   // Set the value
   pTLVx01->mStorageType = (eQMIWMSStorageTypes)uStorageType;
   offset += tlvx01Sz;

   // Add the uMode
  // Check size
    WORD tlvx10Sz = sizeof( sWMSGetStorageMaxSizeRequest_MessageMode );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx10Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x10;
    pHeader->mLength = tlvx10Sz;

    offset += sizeof( sQMIRawContentHeader );

    // The SPC
```

```
    sWMSGetStorageMaxSizeRequest_MessageMode * pTLVx10;
    pTLVx10 = (sWMSGetStorageMaxSizeRequest_MessageMode*)(pOut + offset);
    memset( pTLVx10, 0, tlvx10Sz );

    // Set the values
    pTLVx10->mMode = (eQMIWMSMessageProtocols)uMode;

    offset += tlvx10Sz;

  *pOutLen = offset;
  return eGOBI_ERR_NONE;
}

ULONG ParseGetSmsMaxSize(
  ULONG              inLen,
  const BYTE *    pIn,
  int *           pSmsMaxSize ,
  int *           pSmsUsedSize)
{
  // Validate arguments
  if (pIn == 0 )
  {
    return eGOBI_ERR_INVALID_ARG;
  }

  // Find the TLV
  const sWMSGetStorageMaxSizeResponse_MaxSize * pTLVx01;
  ULONG outLenx01;
  ULONG rc = GetTLV( inLen, pIn, 0x01, &outLenx01, (const BYTE **)&pTLVx01 );
  if (rc != eGOBI_ERR_NONE)
  {
    return rc;
  }

  // Is the TLV large enough?
  if (outLenx01 < sizeof( sWMSGetStorageMaxSizeResponse_MaxSize ))
  {
    return eGOBI_ERR_MALFORMED_RSP;
  }

  *pSmsMaxSize = pTLVx01->mMaxStorageSizeInMessages;

  const sWMSGetStorageMaxSizeResponse_AvailableSize * pTLVx10;
  ULONG outLenx10;
  rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
  if (rc != eGOBI_ERR_NONE)
  {
    return rc;
  }

  // Is the TLV large enough?
  if (outLenx10 < sizeof( sWMSGetStorageMaxSizeResponse_AvailableSize ))
  {
    return eGOBI_ERR_MALFORMED_RSP;
  }
```

```
    *pSmsUsedSize = pTLVx01->mMaxStorageSizeInMessages-pTLVx10->mFreeStorageSizeInMessages;

    return eGOBI_ERR_NONE;
}

ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };

    PackGetSmsMaxSize(&li,req, eQMIWMSStorageTypes_NV, eQMIWMSMessageProtocols_GSMWCDMA
Unsupported);
    status = WMSGetStorageMaxSize ( mhGobi, 2000, li, &req[0], &lo, &rsp[0] );
    if (status != 0)
    {
        return status;
    }

    status = ParseGetSmsMaxSize( lo,&rsp[0],pSmsMaxSize,pSmsUsedSize);
```

3.1.12. WMSGetSMSCAddress

Description	Queries the currently configured SMSC address
Request Structure	

Response Structure	// Structure to describe response TLV 0x01 for WMSGetSMSCAddress()
	struct sWMSGetSMSCAddressResponse_Address
	{
	char mSMSCAddressType[3];
	UINT8 mSMSCAddressLength;
	
	// This array must be the size specified by mSMSCAddressLength
	// char mSMSCAddress[1];
	};

Return Value	QMI_ERR_NONE — No error in the request QMI_ERR_INTERNAL — Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG — Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_OP_DEVICE_UNSUPPORTED — Selected operation is not supported by the device QMI_ERR_DEVICE_NOT_READY — Device has not yet read this value QMI_ERR_NOT_PROVISIONED — Device does not have this value provisioned

Constraint	

Example:

```
ULONG ParseGetSMSCAddress(
    ULONG                 inLen,
    const BYTE *          pIn,
    BYTE                  addressSize,
    CHAR *                pSMSCAddress,
    BYTE                  typeSize,
    CHAR *                pSMSCType )
{
    // Validate arguments
    if (pIn == 0
    || addressSize == 0 || pSMSCAddress == 0
    || typeSize == 0 || pSMSCType == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Assume empty
    pSMSCAddress[0] = 0;
    pSMSCType[0] = 0;

    // Get the address (mandatory)
    const sWMSGetSMSCAddressResponse_Address * pTLVx01;
    ULONG outLenx01;
    ULONG rc = GetTLV( inLen, pIn, 0x01, &outLenx01, (const BYTE **)&pTLVx01 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
    }

    if (outLenx01 < sizeof( sWMSRawSendResponse_CauseCode ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    // Handle the type as a string (maximum 3 chars)
    std::string smscType( &pTLVx01->mSMSCAddressType[0], 3 );

    // Is the SMSC type present? (optional)
    ULONG smscTypeLen = (ULONG)smscType.size();
    if (smscTypeLen > 0)
    {
        // Space to perform copy?
        if (typeSize < smscTypeLen + 1)
        {
            return eGOBI_ERR_BUFFER_SZ;
        }

        memcpy( pSMSCType, &pTLVx01->mSMSCAddressType[0], smscTypeLen );
        pSMSCType[smscTypeLen] = 0;
    }

    // Treat the address as a null terminated string
```

```
    std::string smscAddr( (const CHAR *)pTLVx01
                          + sizeof( sWMSGetSMSCAddressResponse_Address ),
                          pTLVx01->mSMSCAddressLength );

    ULONG smscAddrLen = (ULONG)smscAddr.size();
    if (addressSize < smscAddrLen + 1)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    memcpy( pSMSCAddress, smscAddr.c_str(), addressSize );
    pSMSCAddress[addressSize] = 0;

    return rc;
}

ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };

    status = WMSGetSMSCAddress ( mhGobi, 2000, 0,0,&lo,&rsp[0]);
    if (status != 0)
    {
        return status;
    }
    ParseGetSMSCAddress(lo,rsp,NumberSize,pAddNum,typeSize,pAddType);
```

3.1.13. WMSSetMemoryStatus

Description	Indicates whether the client has storage available for new SMS messages

Request Structure	// Structure to describe request TLV 0x01 for WMSSetMemoryStatus()
	struct sWMSSetMemoryStatusRequest_Status
	{
	INT8 mMemoryIsAvailable;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_MISSING_ARG A required TLV was not provided

	QMI_ERR_INVALID_ARG One of the parameters specified contains an invalid value
Constraint	

Example:

```
UINT8 req[256] = { 0 };
    UINT8 * pData = (UINT8 *)&req[0];

    sQMIRawContentHeader * pTLV = (sQMIRawContentHeader *)pData;
    pTLV->mTypeID = 0x01;
    pTLV->mLength = (UINT16)sizeof( sWMSSetMemoryStatusRequest_Status );
    pData += sizeof( sQMIRawContentHeader );

    sWMSSetMemoryStatusRequest_Status * pID =
        (sWMSSetMemoryStatusRequest_Status *)pData;

    pID->mMemoryIsAvailable = 0;

    pData += sizeof( sWMSSetMemoryStatusRequest_Status );

    // Cancel the request with the device
    ULONG li = (ULONG)pData - (ULONG)&req[0];
    status = WMSSetMemoryStatus ( mhGobi, 2000, li, &req[0], 0, 0 );
```

3.1.14. WMSSetPrimaryClient

Description	Allows the client to set or unset itself as the primary client of QMI_WMS

Request Structure	// Structure to describe request TLV 0x01 for WMSSetPrimaryClient()
	struct sWMSSetPrimaryClientRequest_PrimaryClientInfo
	{
	INT8 mPrimaryClient;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_MISSING_ARG A required TLV was not provided QMI_ERR_INVALID_ARG One of the parameters specified contains an invalid value

Constraint	

Example:

```
UINT8 req[256] = { 0 };
    UINT8 * pData = (UINT8 *)&req[0];

    sQMIRawContentHeader * pTLV = (sQMIRawContentHeader *)pData;
    pTLV->mTypeID = 0x01;
    pTLV->mLength = (UINT16)sizeof( sWMSSetPrimaryClientRequest_PrimaryClientInfo );
    pData += sizeof( sQMIRawContentHeader );

    sWMSSetPrimaryClientRequest_PrimaryClientInfo * pID =
        (sWMSSetPrimaryClientRequest_PrimaryClientInfo *)pData;
    pID->mPrimaryClient = primary_client;

    pData += sizeof( sWMSSetPrimaryClientRequest_PrimaryClientInfo );

    // Cancel the request with the device
    ULONG li = (ULONG)pData - (ULONG)&req[0];
    status = WMSSetPrimaryClient ( mhGobi, 2000, li, &req[0], 0, 0 );
```

3.1.15. PBMSetIndicationRegistrationState

Description	Sets the registration state for different QMI_PBM indications for the requesting control point

Request Structure	// Structure to describe request TLV 0x01 for PBMSetIndicationRegistrationState()
	struct sPBMSetIndicationRegistrationStateRequest_Mask
	{
	bool mRecordUpdate:1;
	bool mPhonebookReady:1;
	bool mEmergencyNumberList:1;
	bool mHiddenRecordStatus:1;
	bool mAASUpdate:1;
	bool mGASUpdate:1;
	
	// Padding out 26 bits
	UINT8 mReserved1:2;
	UINT8 mReserved2[3];
	};

Response Structure	// Structure to describe response TLV 0x10 for PBMSetIndicationRegistrationState()
	struct sPBMSetIndicationRegistrationStateResponse_Mask
	{
	bool mRecordUpdate:1;
	bool mPhonebookReady:1;
	bool mEmergencyNumberList:1;

	bool mHiddenRecordStatus:1;
	bool mAASUpdate:1;
	bool mGASUpdate:1;
	
	// Padding out 26 bits
	UINT8 mReserved1:2;
	UINT8 mReserved2[3];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	If the control point registers for : • Record Update events – QMI_PBM_RECORD_UPDATE_IND is received whenever there is a change (add/edit/delete) in any of the PB records. • Phonebook Ready events – PB Ready indications via QMI_PBM_PB_READY_IND are received. If all the PBs in a session are ready by the time the control point registers, QMI_PBM_ALL_PB_INIT_DONE is sent soon after the registration to make up for the PB Ready indications it missed because of late registration. • Emergency Number List events – Notification of a change in emergency numbers through QMI_PBM_EMERGENCY_LIST_IND. Whenever there is a change in the registration status from disabled to enabled, the control point receives the list of emergency numbers applicable at that time. • Hidden Record Status events – QMI_PBM_HIDDEN_RECORD_STATUS_IND is sent during the phone power-up and whenever there is a change in the hidden status. • Additional number Alpha String Update events – QMI_PBM_AAS_UPDATE_IND is sent whenever there is a change (add/edit/delete) in any of the AAS items. • Grouping information Alpha String Update events – QMI_PBM_GAS_UPDATE_IND is sent whenever there is a change (add/edit/delete) in any of the GAS items.

MSG ID		
	QMI_PBM_RECORD_UPDATE_IND	0x0009
	QMI_PBM_PB_READY_IND	0x000B
	QMI_PBM_ALL_PB_INIT_DONE _IND	0x000D
	QMI_PBM_EMERGENCY_LIST_IND	0x000C
	QMI_PBM_HIDDEN_RECORD_STATUS_IND	0x0013
	QMI_PBM_AAS_UPDATE_IND	0x0018
	QMI_PBM_GAS_UPDATE_IND	0x0019

Example:

```
ULONG PackPbSetIndReg(
   ULONG *                          pOutLen,
    BYTE *                          pOut)
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sPBMSetIndicationRegistrationStateRequest_Mask );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sPBMSetIndicationRegistrationStateRequest_Mask * pTLVx01;
    pTLVx01 = (sPBMSetIndicationRegistrationStateRequest_Mask*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the value
    pTLVx01->mRecordUpdate = true;
    pTLVx01->mPhonebookReady = true;

    offset += tlvx01Sz;

    *pOutLen = offset;
    return eGOBI_ERR_NONE;
}

ULONG cGobiCMDLL::SetPBMAllInitDoneCB( tFNGenericCallback pCallback )
{
    ULONG status = 1;
    if( mpFnSetGenericCallback == 0 || mhGobi == 0)
    {
        return status;
    }
```

```
    // Configure the callback with the API
    status = mpFnSetGenericCallback( mhGobi, 12, 13, pCallback );
    return status;
}


void __cdecl PBMAllInitDoneCallback(
    ULONG                                svcID,
    ULONG                                msgID,
    GOBIHANDLE                            /* handle */,
    ULONG                                outLen,
    const BYTE *                        pOut )
{
    if (svcID != 12 || msgID != 13)
    {
        return;
    }

    ULONG state = ULONG_MAX;
    ULONG cer = ULONG_MAX;


    std::map <UINT8, const sQMIRawContentHeader *> tlvs = GetTLVs( &pOut[0], outLen );
    std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x01 );
    if (pIter != tlvs.end())
    {
        const sQMIRawContentHeader * pTmp = pIter->second;
        if (pTmp->mLength >= sizeof (sPBMAllReadyIndication_Info))
        {
            pTmp++;
            const sPBMAllReadyIndication_Info * pState =
                (const sPBMAllReadyIndication_Info *)pTmp;

                    ULONG status = 1;
                    status = mGobiCMDLL.GetPbCapability(&g_VolumnPBInSim,&g_ActualReadPbSize,&g_
PBNumMaxLen,&g_PBNameMaxLen);
                    if(status != 0)
                    {
                            OutputDebugString("GetPbCapability error!!!.");
                    }
                    else
                    {
                            pTheDialog->PostMessage(WM_PB_READ_START, 0, 0);
                    }
        }
    }

}


ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };
```

```
    PackPbSetIndReg(&li,req);
    status = mpFnPbSetIndReg( mhGobi, 2000, li, &req[0], &lo, &rsp[0] );
SetPBMAllInitDoneCB(PBMAllInitDoneCallback);
```

3.1.16. PBMGetCapabilities

Description	Returns the capabilities of the PB requested

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PBMGetCapabilities()
	struct sPBMGetCapabilitiesRequest_Info
	{
	eQMIPBMSessionTypes mSessionType;
	eQMIPBMPhonebookTypes mPhonebookType;
	};

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for PBMGetCapabilities()
	struct sPBMGetCapabilitiesResponse_Basic
	{
	eQMIPBMSessionTypes mSessionType;
	eQMIPBMPhonebookTypes mPhonebookType;
	UINT16 mRecordsUsed;
	UINT16 mMaximumRecords;
	UINT8 mMaximumNumberLength;
	UINT8 mMaximumNameLength;
	};
	
	// Structure to describe response TLV 0x11 for PBMGetCapabilities()
	struct sPBMGetCapabilitiesResponse_Group
	{
	UINT8 mMaximumGroupsPossible;
	UINT8 mMaximumGroupTagLength;
	};
	
	// Structure to describe response TLV 0x12 for PBMGetCapabilities()
	struct sPBMGetCapabilitiesResponse_AdditionalNumber
	{
	UINT8 mMaximumAdditionalNumbersPossible;
	UINT8 mMaximumAdditionalNumberLength;

```
    UINT8 mMaximumAdditionalNumberTagLength;
};

// Structure to describe response TLV 0x13 for PBMGetCapabilities()
struct sPBMGetCapabilitiesResponse_Email
{
    UINT8 mMaximumEmailsPossible;
    UINT8 mMaximumEmailAddressLength;
};

// Structure to describe response TLV 0x14 for PBMGetCapabilities()
struct sPBMGetCapabilitiesResponse_SecondName
{
    UINT8 mMaximumSecondNameLength;
};

// Structure to describe response TLV 0x15 for PBMGetCapabilities()
struct sPBMGetCapabilitiesResponse_HiddenRecords
{
    INT8 mHiddenEntrySupported;
};

// Structure to describe response TLV 0x16 for PBMGetCapabilities()
struct sPBMGetCapabilitiesResponse_GAS
{
    UINT8 mMaximumGASStringLength;
};

// Structure to describe response TLV 0x17 for PBMGetCapabilities()
struct sPBMGetCapabilitiesResponse_AAS
{
    UINT8 mMaximumAASStringLength;
};

// Structure to describe response TLV 0x18 for PBMGetCapabilities()
struct sPBMGetCapabilitiesResponse_Protection
{
```

	eQMIPBMProtectionMethods mProtectionMethod;
	};
	
	// Structure to describe response TLV 0x19 for PBMGetCapabilities()
	struct sPBMGetCapabilitiesResponse_Sets
	{
	UINT16 mNumberOfPhonebookSets;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_ARG_TOO_LONG More than the maximum allowed thresholds were specified QMI_ERR_INVALID_SESSION_TYPE Invalid session type was provided in the request QMI_ERR_INVALID_PB_TYPE Invalid phonebook type was provided in the request QMI_ERR_NO_SIM SIM is not present QMI_ERR_PB_NOT_READY Phonebook is not ready to be accessed QMI_ERR_PIN_RESTRICTION Phonebook access is restricted by a PIN QMI_ERR_PUK_RESTRICTION Phonebook access is restricted by a PUK (Personal Unblocking Key) QMI_ERR_PB_ACCESS_RESTRICTED Phonebook access is restricted (e.g., ADN access is restricted when FDN check is enabled)
Constraint	

Example:

```
ULONG PackGetCatability(
  ULONG *                       pOutLen,
   BYTE *                       pOut,
   eQMIPBMSessionTypes          inSessionTypes ,
   eQMIPBMPhonebookTypes            inPbTypes )
{
   // Validate arguments
   if (pOut == 0)
   {
      return eGOBI_ERR_INVALID_ARG;
   }

   // Check size
   WORD tlvx01Sz = sizeof( sPBMGetCapabilitiesRequest_Info );
   if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
   {
      return eGOBI_ERR_BUFFER_SZ;
```

```
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sPBMGetCapabilitiesRequest_Info * pTLVx01;
    pTLVx01 = (sPBMGetCapabilitiesRequest_Info*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the value
    pTLVx01->mSessionType = inSessionTypes;
    pTLVx01->mPhonebookType = inPbTypes;

    offset += tlvx01Sz;

    *pOutLen = offset;
    return eGOBI_ERR_NONE;
}

ULONG ParseGetCatability(
    ULONG               inLen,
    const BYTE *        pIn,
    int *               pbMaxRecords ,
    int *               pbUsedRecords,
    int *               pbMaxNumLen,
        int *               pbMaxNameLen)
{
    // Validate arguments
    if (pIn == 0 )
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Find the TLV
    const sPBMGetCapabilitiesResponse_Basic * pTLVx10;
    ULONG outLenx10;
    ULONG rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
    }

    // Is the TLV large enough?
    if (outLenx10 < sizeof( sPBMGetCapabilitiesResponse_Basic ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    *pbMaxRecords = pTLVx10->mMaximumRecords;
    *pbUsedRecords = pTLVx10->mRecordsUsed;
    *pbMaxNumLen = pTLVx10->mMaximumNumberLength;
    *pbMaxNameLen = pTLVx10->mMaximumNameLength;
```

```
    return eGOBI_ERR_NONE;
}

ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };

    PackGetCatability(&li,req,eQMIPBMSessionTypes_GlobalPhonebookOnSlot1,eQMIPBMPhonebookTypes
_AbbreviatedDialingNumber);
    status = PBMGetCapabilities ( mhGobi, 2000, li, &req[0], &lo, &rsp[0] );
    if (status != 0)
    {
        return status;
    }

    status = ParseGetCatability( lo,
                                        &rsp[0],pbMaxRecords,pbUsedRecords,pbMaxNumLen,pbMaxNam
eLen);
```

3.1.17. PBMReadRecords

Description	Initiates the Record Read operation by specifying the range of the records to be read

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PBMReadRecords()
	struct sPBMReadRecordsRequest_Info
	{
	eQMIPBMSessionTypes mSessionType;
	eQMIPBMPhonebookTypes mPhonebookType;
	UINT16 mStartingRecordID;
	UINT16 mEndingRecordID;
	};

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for PBMReadRecords()
	struct sPBMReadRecordsResponse_RecordsRead
	{
	UINT16 mNumberOfRecords;
	};

IND Structure	**Mandatory TLVs**
	// Structure to describe indication TLV 0x01 for PBM ReadRecordsIndication
	struct sPBMReadRecordsIndication_Basic
	{

```
    UINT16 mSequenceNumber;
    eQMIPBMSessionTypes mSessionType;
    eQMIPBMPhonebookTypes mPhonebookType;
    UINT8 mNumberOfRecords;

    struct sRecord1
    {
        UINT16 mRecordID;
        eQMIPBMNumberTypes mNumberType;
        eQMIPBMNumberPlans mNumberPlan;
        UINT8 mNumberLength;

        // This array must be the size specified by mNumberLength
        // char mNumber[1];
    };

    struct sRecord2
    {
        UINT8 mNameLength;

        // This array must be the size (in BYTEs) specified by mNameLength
        // wchar_t mName[1];
    };

    struct sRecord
    {
        sRecord1 mRecord1;
        sRecord2 mRecord2;
    };

    // This array must be the size specified by mNumberOfRecords
    // sRecord mRecords[1];
};
```

Optional TLVs

```
// Structure to describe indication TLV 0x10 for PBM ReadRecordsIndication
struct sPBMReadRecordsIndication_SecondName
{
```

```
    UINT8 mNumberOfRecords;

    struct sRecord
    {
        UINT16 mRecordID;
        UINT8 mSecondNameLength;

        // This array must be the size (in BYTEs) specified by mSecondNameLength
        // wchar_t mSecondName[1];
    };

    // This array must be the size specified by mNumberOfRecords
    // sRecord mRecords[1];
};

// Structure to describe indication TLV 0x11 for PBM ReadRecordsIndication
struct sPBMReadRecordsIndication_AdditionalNumber
{
    UINT8 mNumberOfRecords;

    struct sRecord
    {
        UINT16 mRecordID;
        UINT8 mAdditionalNumberCount;

        struct sAdditionalNumber1
        {
            eQMIPBMNumberTypes mNumberType;
            eQMIPBMNumberPlans mNumberPlan;
            UINT8 mNumberLength;

            // This array must be the size specified by mNumberLength
            // char mNumber[1];
        };

        struct sAdditionalNumber2
        {
```

```
            UINT8 mTagID;
        };

        struct sAdditionalNumber
        {
            sAdditionalNumber1 mAdditionalNumber1;
            sAdditionalNumber2 mAdditionalNumber2;
        };

        // This array must be the size specified by mAdditionalNumberCount
        // sAdditionalNumber mAdditionalNumbers[1];
    };

    // This array must be the size specified by mNumberOfRecords
    // sRecord mRecords[1];
};

// Structure to describe indication TLV 0x12 for PBM ReadRecordsIndication
struct sPBMReadRecordsIndication_Group
{
    UINT8 mNumberOfRecords;

    struct sRecord
    {
        UINT16 mRecordID;
        UINT8 mGroupCount;

        // This array must be the size specified by mGroupCount
        // UINT8 mGroupID[1];
    };

    // This array must be the size specified by mNumberOfRecords
    // sRecord mRecords[1];
};

// Structure to describe indication TLV 0x13 for PBM ReadRecordsIndication
struct sPBMReadRecordsIndication_Email
```

```
{
    UINT8 mNumberOfRecords;

    struct sRecord
    {
        UINT16 mRecordID;
        UINT8 mEmailCount;

        struct sEmail
        {
            UINT8 mAddressLength;

            // This array must be the size (in BYTEs) specified by mAddressLength
            // wchar_t mAddress[1];
        };

        // This array must be the size specified by mEmailCount
        // sEmail mEmails[1];
    };

    // This array must be the size specified by mNumberOfRecords
    // sRecord mRecords[1];
};

// Structure to describe indication TLV 0x14 for PBM ReadRecordsIndication
struct sPBMReadRecordsIndication_Hidden
{
    UINT8 mNumberOfRecords;

    struct sRecord
    {
        UINT16 mRecordID;
        INT8 mHidden;
    };

    // This array must be the size specified by mNumberOfRecords
    // sRecord mRecords[1];
```

	};
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_INVALID_ID Record ID in the request is not valid QMI_ERR_INVALID_SESSION_TYPE Invalid session type was provided in the request QMI_ERR_INVALID_PB_TYPE Invalid phonebook type was provided in the request QMI_ERR_NO_SIM SIM is not present QMI_ERR_PB_NOT_READY Phonebook is not ready to be accessed QMI_ERR_INVAILD_ARG Invalid combination of Start and End Record IDs QMI_ERR_PIN_RESTRICTION Phonebook access is restricted by a PIN QMI_ERR_PUK_RESTRICTION Phonebook access is restricted by a PUK QMI_ERR_PB_ACCESS_RESTRICTED Phonebook access is restricted (e.g., ADN access is restricted when FDN check is enabled) QMI_ERR_PB_DELETE_IN_PROG Records in the phonebook are being deleted; phonebook access during delete operations is rejected to avoid unexpected results
Constraint	If the result code is QMI_RESULT_SUCCESS, the records data is returned in the subsequent QMI_PBM_RECORD_READ_IND
MSG ID	QMI_PBM_RECORD_READ_IND 0x0004

Example:

```
ULONG cGobiCMDLL::SetPBReadRecordIndCB( tFNGenericCallback pCallback )
{
    ULONG status = 1;
    if( mpFnSetGenericCallback == 0 || mhGobi == 0)
    {
        return status;
    }

    // Configure the callback with the API
    status = mpFnSetGenericCallback( mhGobi, 12, 4, pCallback );
    return status;
}

void __cdecl PBMReadRecordsIndCallback(
    ULONG                           svcID,
    ULONG                           msgID,
    GOBIHANDLE                       /* handle */,
    ULONG                           outLen,
    const BYTE *                    pOut )
{
```

```
    if (svcID != 12 || msgID != 4)
    {
        return;
    }

    ULONG state = ULONG_MAX;
    ULONG cer = ULONG_MAX;


    std::map <UINT8, const sQMIRawContentHeader *> tlvs = GetTLVs( &pOut[0], outLen );
    std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x01 );
    if (pIter != tlvs.end())
    {
        const sQMIRawContentHeader * pTmp = pIter->second;
        if (pTmp->mLength >= sizeof (sPBMReadRecordsIndication_Basic))
        {
            pTmp++;
            const sPBMReadRecordsIndication_Basic * pState =
                (const sPBMReadRecordsIndication_Basic *)pTmp;

                    int offset = sizeof(sPBMReadRecordsIndication_Basic);
                    sRecord1 *pRecord1 = (sRecord1 *)((BYTE *)pState + offset);
                    if(g_pPbOnPC == NULL)
                    {
                            OutputDebugString("PBMReadRecordsIndCallback error!!!.");
                    }


                    int RecordsNum=0;
                    int id = 0;
                    for(RecordsNum=0;RecordsNum<pState->mNumberOfRecords;RecordsNum++)
                    {
                            while((( 0 != strcmp( (char*)g_pPbOnPC[id].number, "" ) )||( 0 != strcmp(
(char*)g_pPbOnPC[id].name, "" ))
                                    ||( 0 != strcmp( (char*)g_pPbOnPC[id].office_no, "" ))||( 0 != strcmp
( (char*)g_pPbOnPC[id].family_no, "" ))
                                    ||( 0 != strcmp( (char*)g_pPbOnPC[id].fax_no, "" )))&&( id < g_Vol
umnPBInSim-1 )))
                                    id ++;

                            memset(&g_pPbOnPC[id],0x00,sizeof(PBITEM));
                            g_pPbOnPC[id].index = pRecord1->mRecordID;
                            if(eQMIPBMNumberTypes_International ==  pRecord1->mNumberType)
                            {
                                    g_pPbOnPC[id].number_type = 145;
                            }
                            else
                            {
                                    g_pPbOnPC[id].number_type = 129;
                            }
                            offset += sizeof(UINT16)+sizeof(eQMIPBMNumberTypes)+sizeof(eQMIPBMN
umberPlans)+sizeof(UINT8);//replace sizeof(sRecord1)
                            memcpy( g_pPbOnPC[id].number,((BYTE *)pState + offset), pRecord1->mNu
mberLength);
                            WCHAR wName[20]={0};
```

```
                        offset +=   pRecord1->mNumberLength;

                        sRecord2 *pRecord2 = (sRecord2 *)((BYTE *)pState + offset);
                        offset += sizeof(sRecord2);
                        memcpy( wName,((BYTE *)pState + offset),pRecord2->mNameLength);
                        WideCharToMultiByte(CP_ACP,NULL,wName,sizeof(wName),(LPSTR)g_pPbO
nPC[id].name,sizeof(g_pPbOnPC[id].name),NULL,NULL);
                        g_ActualPbSize++;
                        if(g_ActualPbSize == g_ActualReadPbSize)
                        {
                                g_bInitPBFinish = TRUE; // read pb items finished
                                g_bInitFinish = TRUE;//TODO limingjun,maybe wait sms init ready
                                pTheDialog->PostMessage(WM_SMS_READ_START, 0, 0);
                                break;
                        }
                        offset += pRecord2->mNameLength;
                        pRecord1 = (sRecord1 *)((BYTE *)pState + offset);
                }
        }
    }

}

ULONG PackPbReadRecords(
  ULONG *                       pOutLen,
    BYTE *                      pOut)
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sPBMReadRecordsRequest_Info );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sPBMReadRecordsRequest_Info * pTLVx01;
    pTLVx01 = (sPBMReadRecordsRequest_Info*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the value
    pTLVx01->mSessionType = eQMIPBMSessionTypes_GlobalPhonebookOnSlot1;
    pTLVx01->mPhonebookType = eQMIPBMPhonebookTypes_AbbreviatedDialingNumber;
    pTLVx01->mStartingRecordID = 1;
    pTLVx01->mEndingRecordID = g_VolumnPBInSim;
```

```
    offset += tlvx01Sz;

    *pOutLen = offset;
    return eGOBI_ERR_NONE;
}


SetPBReadRecordIndCB(PBMReadRecordsIndCallback);

    ULONG li = 1024;
    BYTE req[1024] = { 0 };

    PackPbReadRecords(&li,req);
    status = PBMReadRecords ( mhGobi, 2000, li, &req[0],0,0 );
```

3.1.18. PBMWriteRecord

Description	Adds a new record or modifies an existing record

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PBMWriteRecord()
	struct sPBMWriteRecordRequest_Info1
	{
	eQMIPBMSessionTypes mSessionType;
	eQMIPBMPhonebookTypes mPhonebookType;
	UINT16 mRecordID;
	eQMIPBMNumberTypes mNumberType;
	eQMIPBMNumberPlans mNumberPlan;
	UINT8 mNumberLength;
	
	// This array must be the size specified by mNumberLength
	// char mNumber[1];
	};
	Optional TLVs
	struct sPBMWriteRecordRequest_Info2
	{
	UINT8 mNameLength;
	
	// This array must be the size (in BYTEs) specified by mNameLength
	// wchar_t mName[1];
	};

```
struct sPBMWriteRecordRequest_Info
{
    sPBMWriteRecordRequest_Info1 mPBMWriteRecordRequest_Info1;
    sPBMWriteRecordRequest_Info2 mPBMWriteRecordRequest_Info2;
};

// Structure to describe request TLV 0x10 for PBMWriteRecord()
struct sPBMWriteRecordRequest_SecondName
{
    UINT8 mNameLength;

    // This array must be the size (in BYTEs) specified by mNameLength
    // wchar_t mName[1];
};

// Structure to describe request TLV 0x11 for PBMWriteRecord()
struct sPBMWriteRecordRequest_AdditionalNumber
{
    UINT8 mAdditionalNumberCount;

    struct sAdditionalNumber1
    {
        eQMIPBMNumberTypes mNumberType;
        eQMIPBMNumberPlans mNumberPlan;
        UINT8 mNumberLength;

        // This array must be the size specified by mNumberLength
        // char mNumber[1];
    };

    struct sAdditionalNumber2
    {
        UINT8 mTagID;
    };

    struct sAdditionalNumber
    {
```

```
        sAdditionalNumber1 mAdditionalNumber1;
        sAdditionalNumber2 mAdditionalNumber2;
    };

    // This array must be the size specified by mAdditionalNumberCount
    // sAdditionalNumber mAdditionalNumbers[1];
};

// Structure to describe request TLV 0x12 for PBMWriteRecord()
struct sPBMWriteRecordRequest_Group
{
    UINT8 mGroupCount;

    // This array must be the size specified by mGroupCount
    // UINT8 mGroupID[1];
};

// Structure to describe request TLV 0x13 for PBMWriteRecord()
struct sPBMWriteRecordRequest_Email
{
    UINT8 mEmailCount;

    struct sEmail
    {
        UINT8 mAddressLength;

        // This array must be the size (in BYTEs) specified by mAddressLength
        // wchar_t mAddress[1];
    };

    // This array must be the size specified by mEmailCount
    // sEmail mEmails[1];
};

// Structure to describe request TLV 0x14 for PBMWriteRecord()
struct sPBMWriteRecordRequest_Hidden
{
```

	INT8 mHidden; };

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for PBMWriteRecord()
	struct sPBMWriteRecordResponse_Info
	{
	UINT16 mRecordID;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_INVALID_ID Record ID in the request is not valid QMI_ERR_INVALID_SESSION_TYPE Invalid session type was provided in the request QMI_ERR_INVALID_PB_TYPE Invalid phonebook type was provided in the request QMI_ERR_NO_SIM SIM is not present QMI_ERR_NUMBER_TOO_LONG Number sent in the request is longer than expected QMI_ERR_PB_NOT_READY Phonebook is not ready to be accessed QMI_ERR_TEXT_TOO_LONG Name text provided in the request is longer than expected QMI_ERR_PIN_RESTRICTION Phonebook access is restricted by a PIN QMI_ERR_PIN2_RESTRICTION Phonebook access is restricted by a PIN2 QMI_ERR_PUK_RESTRICTION Phonebook access is restricted by a PUK QMI_ERR_PUK2_RESTRICTION Phonebook access is restricted by a PUK2 QMI_ERR_PB_ACCESS_RESTRICTED Phonebook access is restricted (e.g., ADN access is restricted when FDN check is enabled) QMI_ERR_PB_DELETE_IN_PROG Records in the phonebook are being deleted; phonebook access during delete operations is rejected to avoid unexpected results QMI_ERR_DEVICE_MEMORY_ ERROR Write operation failed in SIM card due to memory problems
Constraint	

Example:

```
ULONG  PackPbWriteRecords(
   ULONG  *                          pOutLen,
    BYTE  *                          pOut,
    int  inNumType,char  *  chName,char  *  chNumber,int  inIndex)
```

```
{
    // Validate arguments
    if (pOut == 0|| chName == 0 || chNumber == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }
    // Add arguments
    std::wstring sName( (WCHAR *)chName );
    ULONG sNameSz = (ULONG)sName.size()*2;
    std::string sNum( chNumber );
    ULONG sNumSz = (ULONG)sNum.size();

    // Check size
    WORD tlvx01Sz = sizeof( sPBMWriteRecordRequest_Info )+(WORD)sNameSz+(WORD)sNumSz;
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    // First part of the TLV
    sPBMWriteRecordRequest_Info1 * pTLVx01_1;
    pTLVx01_1 = (sPBMWriteRecordRequest_Info1*)(pOut + offset);
    memset( pTLVx01_1, 0, tlvx01Sz );

    pTLVx01_1->mSessionType = eQMIPBMSessionTypes_GlobalPhonebookOnSlot1;
    pTLVx01_1->mPhonebookType = eQMIPBMPhonebookTypes_AbbreviatedDialingNumber;
     pTLVx01_1->mRecordID = inIndex;//0:add new record

    if(145 == inNumType)
    {
            pTLVx01_1->mNumberType = eQMIPBMNumberTypes_International;
    }
    else
    {
            pTLVx01_1->mNumberType = eQMIPBMNumberTypes_National;
    }
    pTLVx01_1->mNumberPlan = eQMIPBMNumberPlans_ISDN;
    pTLVx01_1->mNumberLength = sNumSz;

    offset += sizeof( sPBMWriteRecordRequest_Info1 );

    // mOldPINValue string
    memcpy( (pOut + offset), sNum.c_str(), sNumSz );
    offset += sNumSz;

    // Second part of the TLV
    sPBMWriteRecordRequest_Info2 * pTLVx01_2;
    pTLVx01_2 = (sPBMWriteRecordRequest_Info2*)(pOut + offset);
```

```
    pTLVx01_2->mNameLength = sNameSz;
    offset += sizeof( sPBMWriteRecordRequest_Info2 );

    // mNewPINValue string
    memcpy( (pOut + offset), sName.c_str(), sNameSz );
    offset += sNameSz;

    *pOutLen = offset;

    return eGOBI_ERR_NONE;
}

ULONG ParsePbWriteRecords(
    ULONG               inLen,
    const BYTE *     pIn,
    int *            pRecordsID)
{
    // Validate arguments
    if (pIn == 0 )
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Find the TLV
    const sPBMWriteRecordResponse_Info * pTLVx10;
    ULONG outLenx10;
    ULONG rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
    }

    // Is the TLV large enough?
    if (outLenx10 < sizeof( sPBMWriteRecordResponse_Info ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    *pRecordsID = pTLVx10->mRecordID;

    return eGOBI_ERR_NONE;
}



    ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 1024;
    BYTE rsp[1024] = { 0 };
    char chUscName[50]={0};

    ::MultiByteToWideChar(CP_ACP,MB_PRECOMPOSED,(LPCCH)pItem->name,strlen((const char*)pItem-
>name),(WCHAR *)chUscName,22);

    PackPbWriteRecords(&li,req,pItem->number_type,chUscName,(char *)pItem->number,0);
```

```
status = PBMWriteRecord( mhGobi,2000,li,&req[0],&lo,&rsp[0] );
if (status != 0)
{
    return status;
}
status = ParsePbWriteRecords(lo,rsp,&pItem->index);
```

3.1.19. PBMDeleteRecord

Description	Deletes a PB record

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PBMDeleteRecord()
	struct sPBMDeleteRecordRequest_Info
	{
	eQMIPBMSessionTypes mSessionType;
	eQMIPBMPhonebookTypes mPhonebookType;
	UINT16 mRecordID;
	};

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for PBMDeleteRecord()
	struct sPBMDeleteRecordResponse_Info
	{
	UINT16 mRecordID;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_INVALID_ID Record ID in the request is not valid QMI_ERR_INVALID_SESSION_TYPE Invalid session type was provided in the request QMI_ERR_INVALID_PB_TYPE Invalid phonebook type was provided in the request QMI_ERR_NO_SIM SIM is not present QMI_ERR_PB_NOT_READY Phonebook is not ready to be accessed QMI_ERR_PIN_RESTRICTION Phonebook access is restricted by a PIN QMI_ERR_PIN2_RESTRICTION Phonebook access is restricted by a PIN2 QMI_ERR_PUK_RESTRICTION Phonebook access is restricted by a PUK QMI_ERR_PUK2_RESTRICTION Phonebook access is restricted by a PUK2

	QMI_ERR_PB_ACCESS_RESTRICTED Phonebook access is restricted (e.g., ADN access is restricted when FDN check is enabled) QMI_ERR_PB_DELETE_IN_PROG Records in the phonebook are being deleted; phonebook access during delete operations is rejected to avoid unexpected results QMI_ERR_DEVICE_MEMORY_ ERROR Write operation failed in SIM card due to memory problems
Constraint	

Example:

```
ULONG PackPbDeleteRecords(ULONG * pOutLen,BYTE * pOut,int inRecordID)
{
        // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sPBMDeleteRecordRequest_Info );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sPBMDeleteRecordRequest_Info * pTLVx01;
    pTLVx01 = (sPBMDeleteRecordRequest_Info*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the value
    pTLVx01->mSessionType = eQMIPBMSessionTypes_GlobalPhonebookOnSlot1;
    pTLVx01->mPhonebookType = eQMIPBMPhonebookTypes_AbbreviatedDialingNumber;
    pTLVx01->mRecordID = inRecordID;

    offset += tlvx01Sz;

    *pOutLen = offset;

    return eGOBI_ERR_NONE;
}
```

```
ULONG li = 1024;
BYTE req[1024] = { 0 };


PackPbDeleteRecords(&li,req,inRecordID);
status = mpFnPbDeleteRecords( mhGobi,2000,li,&req[0],0,0 );
```

3.1.20. NASGetHomeNetwork

Description	Retrieves information about the home network of the device
Request Structure	NULL

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for NASGetHomeNetwork()
	struct sNASGetHomeNetworkResponse_HomeNetwork
	{
	UINT16 mMobileCountryCode;
	UINT16 mMobileNetworkCode;
	UINT8 mDescriptionLength;
	
	// This array must be the size specified by mDescriptionLength
	// char mDescription[1];
	};
	Optional TLVs
	// Structure to describe response TLV 0x10 for NASGetHomeNetwork()
	struct sNASGetHomeNetworkResponse_HomeIDs
	{
	UINT16 mSystemID;
	UINT16 mNetworkID;
	};
	
	// Structure to describe response TLV 0x11 for NASGetHomeNetwork()
	struct sNASGetHomeNetworkResponse_ExtendedHomeNetwork
	{
	UINT16 mMobileCountryCode;
	UINT16 mMobileNetworkCode;
	eQMINASNetworkDescriptionDisplays mDisplayNetworkDescription;
	eQMINASNetworkDescriptionEncodings mNetworkDescriptionEncoding;
	UINT8 mNetworkDescriptionLength;

	// This array must be the size specified by mNetworkDescriptionLength
	// UINT8 mNetworkDescription[1];
	};
	
	// Structure to describe response TLV 0x12 for NASGetHomeNetwork()
	struct sNASGetHomeNetworkResponse_3GPPHomeNetworkMNC
	{
	INT8 mHomeNetworkIs3GPP;
	INT8 mMNCIncludesPCSDigit;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_NOT_PROVISIONED Home network is not provisioned on the device
Constraint	

Example:

```
ULONG ParseGetHomeNetwork(
    ULONG                 inLen,
    const BYTE *          pIn,
    WORD *                pMCC,
    WORD *                pMNC,
    BYTE                  nameSize,
    CHAR *                pName,
    WORD *                pSID,
    WORD *                pNID )
{
    // Validate arguments
    if (pIn == 0
    ||  pMCC == 0
    ||  pMNC == 0
    ||  nameSize == 0
    ||  pName == 0
    ||  pSID == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Assume failure
    *pName = 0;
    *pSID = USHRT_MAX;
    *pNID = USHRT_MAX;

    // Find the name (mandatory)
```

```
const sNASGetHomeNetworkResponse_HomeNetwork * pTLVx01;
ULONG outLenx01;
ULONG rc = GetTLV( inLen, pIn, 0x01, &outLenx01, (const BYTE **)&pTLVx01 );
if (rc != eGOBI_ERR_NONE)
{
    return rc;
}

if (outLenx01 < sizeof( sNASGetHomeNetworkResponse_HomeNetwork ))
{
    return eGOBI_ERR_MALFORMED_RSP;
}

// Populate the variables
*pMCC = pTLVx01->mMobileCountryCode;
*pMNC = pTLVx01->mMobileNetworkCode;

ULONG descLen = pTLVx01->mDescriptionLength;
const CHAR * pDesc;

// Verify there is room for the array in the TLV
if (outLenx01 < sizeof( sNASGetHomeNetworkResponse_HomeNetwork )
                + sizeof( CHAR ) * descLen)
{
    return eGOBI_ERR_MALFORMED_RSP;
}

// Space to perform the copy?
if (nameSize < descLen + 1)
{
    return eGOBI_ERR_BUFFER_SZ;
}

// Align to the first array element
pDesc = (const CHAR *)((const BYTE *)pTLVx01
        + sizeof( sNASGetHomeNetworkResponse_HomeNetwork ));

memcpy( pName, pDesc, descLen );
pName[descLen] = 0;


// Find the SID/NID (optional)
const sNASGetHomeNetworkResponse_HomeIDs * pTLVx10;
ULONG outLenx10;
rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
if (rc == eGOBI_ERR_NONE)
{
    if (outLenx10 < sizeof( sNASGetHomeNetworkResponse_HomeIDs ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    *pSID = pTLVx10->mSystemID;
    *pNID = pTLVx10->mNetworkID;
}
```

```
    return eGOBI_ERR_NONE;
}
    ULONG lo = 8096;
    BYTE rsp[8096] = { 0 };
    status = mpFnGetHomeNetwork( mhGobi, 2000, 0, 0, &lo, &rsp[0] );
    if (status != 0)
    {
        return status;
    }

    status = ParseGetHomeNetwork( lo,
                                  &rsp[0],
                                  pHomeMCC,
                                  pHomeMNC,
                                  homeNameSize,
                                  pHomeName,
                                  pSID,
                                  pNID );
```

3.1.21. NASGetSystemInfo

Description	Provides the system information
Request Structure	NULL

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for NASGetSystemInfo()
	struct sNASGetSystemInfoResponse_CDMAServiceStatusInfo
	{
	eQMINASServiceStatus mServiceStatus;
	eQMINASPreferredDataBath mPreferredDataPath;
	};
	
	// Structure to describe response TLV 0x11 for NASGetSystemInfo()
	struct sNASGetSystemInfoResponse_CDMA1xEVDOServiceStatusInfo
	{
	eQMINASServiceStatus mServiceStatus;
	eQMINASPreferredDataBath mPreferredDataPath;
	};
	
	// Structure to describe response TLV 0x12 for NASGetSystemInfo()
	struct sNASGetSystemInfoResponse_GSMServiceStatusInfo
	{

```
    eQMINASServiceStatus mServiceStatus;
    eQMINASServiceStatus mTrueServiceStatus;
    eQMINASPreferredDataBath mPreferredDataPath;
};

// Structure to describe response TLV 0x13 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_WCDMAServiceStatusInfo
{
    eQMINASServiceStatus mServiceStatus;
    eQMINASServiceStatus mTrueServiceStatus;
    eQMINASPreferredDataBath mPreferredDataPath;
};

// Structure to describe response TLV 0x14 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_LTEServiceStatusInfo
{
    eQMINASServiceStatus mServiceStatus;
    eQMINASServiceStatus mTrueServiceStatus;
    eQMINASPreferredDataBath mPreferredDataPath;
};

// Structure to describe response TLV 0x15 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_CDMASystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mSystemPRLMatchValid;
    eQMINASPRLIndicator mSystemPRLMatch;
    INT8 mPRevInUseValid;
    eQMINASRevision mProtocolRevisionInUse;
    INT8 mBaseStationPRevValid;
```

```
    eQMINASRevision mBaseStationProtocolRevision;
    INT8 mConcurrentServiceSupportedValid;
    eQMINASConcurrentServiceSupported mConcurrentServiceSupported;
    INT8 mCDMASystemIDValid;
    UINT16 mSystemID;
    UINT16 mNetworkID;
    INT8 mBaseStationInfoValid;
    UINT16 mBaseStationID;
    INT32 mLatitude;
    INT32 mLongitude;
    INT8 mPacketZoneValid;
    UINT16 mPacketZone;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
};

// Structure to describe response TLV 0x16 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_CDMA1xEVDOSystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mSystemPRLMatchValid;
    eQMINASPRLIndicator mSystemPRLMatch;
    INT8 mCDMA1xEVDOPersonalityValid;
    eQMINASCDMA1xEVDOPersonality mCDMA1xEVDOPersonality;
    INT8 mCDMA1xEVDOActiveProtocolValid;
    eQMINASCDMA1xEVDOActiveProtocol mCDMA1xEVDOActiveProtocol;
    INT8 mSectorIDValid;
    UINT8 mSectorID[16];
};
```

```
// Structure to describe response TLV 0x17 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_GSMSystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mEGPRSSupportValid;
    eQMINASEGPRSSupport mEGPRSSupport;
    INT8 mDTMSupportValid;
    eQMINASDTMSupport mDTMSupport;
};

// Structure to describe response TLV 0x18 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_WCDMASystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
```

```
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mHighSpeedCallStatusValid;
    eQMINASHighSpeedCallStatus mHighSpeedCallStatus;
    INT8 mHighSpeedServiceIndicationValid;
    eQMINASHighSpeedCallStatus mHighSpeedServiceIndication;
    INT8 mPrimaryScramblingCodeValue;
    UINT16 mPrimaryScramblingCode;
};

// Structure to describe response TLV 0x19 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_LTESystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
```

```
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mTrackingAreaCodeValid;
    UINT16 mTrackingAreaCode;
};

// Structure to describe response TLV 0x1A for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_MoreCDMASystemInfo
{
    UINT16 mGeoSystemIndex;
    UINT16 mRegistrationPeriod;
};

// Structure to describe response TLV 0x1B for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_MoreCDMA1xEVDOSystemInfo
{
    UINT16 mGeoSystemIndex;
};

// Structure to describe response TLV 0x1C for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_MoreGSMSystemInfo
{
    UINT16 mGeoSystemIndex;
    eQMINASCellBroadcastCaps mCellBroadcastCapability;
};

// Structure to describe response TLV 0x1D for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_MoreWCDMASystemInfo
{
    UINT16 mGeoSystemIndex;
    eQMINASCellBroadcastCaps mCellBroadcastCapability;
};

// Structure to describe response TLV 0x1E for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_MoreLTESystemInfo
```

```
{
    UINT16 mGeoSystemIndex;
};

// Structure to describe response TLV 0x1F for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_GSMCallBarring
{
    eQMINASCallBarringStatus mCSCallBarringStatus;
    eQMINASCallBarringStatus mPSCallBarringStatus;
};

// Structure to describe response TLV 0x20 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_WCDMACallBarring
{
    eQMINASCallBarringStatus mCSCallBarringStatus;
    eQMINASCallBarringStatus mPSCallBarringStatus;
};

// Structure to describe response TLV 0x21 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_LTEVoice
{
    INT8 mLTEVoiceSupported;
};

// Structure to describe response TLV 0x22 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_GSMCipher
{
    eQMINASServiceDomains mCipheringOnServiceDomain;
};

// Structure to describe response TLV 0x23 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_WCDMACipher
{
    eQMINASServiceDomains mCipheringOnServiceDomain;
};

// Structure to describe response TLV 0x24 for NASGetSystemInfo()
```

```
struct sNASGetSystemInfoResponse_TDSCDMAServiceStatusInfo
{
    eQMINASServiceStatus mServiceStatus;
    eQMINASServiceStatus mTrueServiceStatus;
    eQMINASPreferredDataBath mPreferredDataPath;
};

// Structure to describe response TLV 0x25 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_TDSCDMASystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mHighSpeedCallStatusValid;
    eQMINASHighSpeedCallStatus mHighSpeedCallStatus;
    INT8 mHighSpeedServiceIndicationValid;
    eQMINASHighSpeedCallStatus mHighSpeedServiceIndication;
    INT8 mCellParameterIDValid;
    UINT16 mCellParameterID;
    INT8 mCellBroadcastCapabilityValid;
    eQMINASCellBroadcastCaps2 mCellBroadcastCapability;
    INT8 mCSBarringStatusValid;
```

```
    eQMINASCallBarringStatus mCSCallBarringStatus;
    INT8 mPSBarringStatusValid;
    eQMINASCallBarringStatus mPSCallBarringStatus;
    INT8 mCipheringValid;
    eQMINASServiceDomains mCipheringOnServiceDomain;
};

// Structure to describe response TLV 0x26 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_EMBMSCoverage
{
    INT8 mEMBMSSupported;
};

// Structure to describe response TLV 0x27 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_SIMRejectInfo
{
    eQMINASSIMRejectStates mSIMRejectInfo;
};

// Structure to describe response TLV 0x28 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_WCDMAEUTRADetection
{
    eQMINASEUTRAStatus mEUTRADetectionStatus;
};

// Structure to describe response TLV 0x29 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_LTEIMSVoice
{
    INT8 mIMSVoiceSupportAvailable;
};

// Structure to describe response TLV 0x2A for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_LTEVoiceDomain
{
    eQMINASLTEVoiceDomains mLTEVoiceDomain;
};
```

```
// Structure to describe response TLV 0x2B for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_CDMARegZoneID
{
    UINT16 mCDMARegZoneID;
};

// Structure to describe response TLV 0x2C for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_GSMRAC
{
    UINT8 mGSMRoutingAreaCode;
};

// Structure to describe response TLV 0x2D for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_WCDMARAC
{
    UINT8 mWCDMARoutingAreaCode;
};

// Structure to describe response TLV 0x2E for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_CDMAResolvedMCC
{
    UINT16 mCDMAMCCResolvedViaSIDLookup;
};

// Structure to describe response TLV 0x2F for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_RegistrationRestriction
{
    eQMINASRegistrationRestrictions mRegistrationRestriction;
};

// Structure to describe response TLV 0x30 for NASGetSystemInfo()
struct sNASGetSystemInfoResponse_TDSCDMARegistrationDomain
{
    eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
};

// Structure to describe response TLV 0x31 for NASGetSystemInfo()
```

	struct sNASGetSystemInfoResponse_LTERegistrationDomain
	{
	eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
	};
	
	// Structure to describe response TLV 0x32 for NASGetSystemInfo()
	struct sNASGetSystemInfoResponse_WCDMARegistrationDomain
	{
	eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
	};
	
	// Structure to describe response TLV 0x33 for NASGetSystemInfo()
	struct sNASGetSystemInfoResponse_GSMRegistrationDomain
	{
	eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
	};
	
	// Structure to describe response TLV 0x34 for NASGetSystemInfo()
	struct sNASGetSystemInfoResponse_EMBMSTraceID
	{
	INT16 mEMBMSTraceID;
	};
	
	// Structure to describe response TLV 0x35 for NASGetSystemInfo()
	struct sNASGetSystemInfoResponse_WCDMACSGInfo
	{
	UINT32 mCSGID;
	UINT8 mNameLength;
	
	// This array must be the size specified by mNameLength
	// wchar_t mName[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response

	QMI_ERR_INFO_UNAVAILABLE Information is not available at this time
Constraint	
MSG ID	**QMI_NAS_SYS_INFO_IND** 0x004E

IND Structure	// Structure to describe indication TLV 0x10 for NAS SystemInfoIndication
	struct sNASSystemInfoIndication_CDMAServiceStatusInfo
	{
	eQMINASServiceStatus mServiceStatus;
	eQMINASPreferredDataBath mPreferredDataPath;
	};
	
	// Structure to describe indication TLV 0x11 for NAS SystemInfoIndication
	struct sNASSystemInfoIndication_CDMA1xEVDOServiceStatusInfo
	{
	eQMINASServiceStatus mServiceStatus;
	eQMINASPreferredDataBath mPreferredDataPath;
	};
	
	// Structure to describe indication TLV 0x12 for NAS SystemInfoIndication
	struct sNASSystemInfoIndication_GSMServiceStatusInfo
	{
	eQMINASServiceStatus mServiceStatus;
	eQMINASServiceStatus mTrueServiceStatus;
	eQMINASPreferredDataBath mPreferredDataPath;
	};
	
	// Structure to describe indication TLV 0x13 for NAS SystemInfoIndication
	struct sNASSystemInfoIndication_WCDMAServiceStatusInfo
	{
	eQMINASServiceStatus mServiceStatus;
	eQMINASServiceStatus mTrueServiceStatus;
	eQMINASPreferredDataBath mPreferredDataPath;
	};
	
	// Structure to describe indication TLV 0x14 for NAS SystemInfoIndication
	struct sNASSystemInfoIndication_LTEServiceStatusInfo
	{
	eQMINASServiceStatus mServiceStatus;

```
    eQMINASServiceStatus mTrueServiceStatus;
    eQMINASPreferredDataBath mPreferredDataPath;
};

// Structure to describe indication TLV 0x15 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_CDMASystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mSystemPRLMatchValid;
    eQMINASPRLIndicator mSystemPRLMatch;
    INT8 mPRevInUseValid;
    eQMINASRevision mProtocolRevisionInUse;
    INT8 mBaseStationPRevValid;
    eQMINASRevision mBaseStationProtocolRevision;
    INT8 mConcurrentServiceSupportedValid;
    eQMINASConcurrentServiceSupported mConcurrentServiceSupported;
    INT8 mCDMASystemIDValid;
    UINT16 mSystemID;
    UINT16 mNetworkID;
    INT8 mBaseStationInfoValid;
    UINT16 mBaseStationID;
    INT32 mLatitude;
    INT32 mLongitude;
    INT8 mPacketZoneValid;
    UINT16 mPacketZone;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
};
```

```
// Structure to describe indication TLV 0x16 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_CDMA1xEVDOSystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mSystemPRLMatchValid;
    eQMINASPRLIndicator mSystemPRLMatch;
    INT8 mCDMA1xEVDOPersonalityValid;
    eQMINASCDMA1xEVDOPersonality mCDMA1xEVDOPersonality;
    INT8 mCDMA1xEVDOActiveProtocolValid;
    eQMINASCDMA1xEVDOActiveProtocol mCDMA1xEVDOActiveProtocol;
    INT8 mSectorIDValid;
    UINT8 mSectorID[16];
};

// Structure to describe indication TLV 0x17 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_GSMSystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
```

```
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mEGPRSSupportValid;
    eQMINASEGPRSSupport mEGPRSSupport;
    INT8 mDTMSupportValid;
    eQMINASDTMSupport mDTMSupport;
};

// Structure to describe indication TLV 0x18 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_WCDMASystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mHighSpeedCallStatusValid;
    eQMINASHighSpeedCallStatus mHighSpeedCallStatus;
    INT8 mHighSpeedServiceIndicationValid;
    eQMINASHighSpeedCallStatus mHighSpeedServiceIndication;
    INT8 mPrimaryScramblingCodeValue;
```

```
    UINT16 mPrimaryScramblingCode;
};

// Structure to describe indication TLV 0x19 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_LTESystemInfo
{
    INT8 mServiceDomainValid;
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mTrackingAreaCodeValid;
    UINT16 mTrackingAreaCode;
};

// Structure to describe indication TLV 0x1A for NAS SystemInfoIndication
struct sNASSystemInfoIndication_MoreCDMASystemInfo
{
    UINT16 mGeoSystemIndex;
    UINT16 mRegistrationPeriod;
};

// Structure to describe indication TLV 0x1B for NAS SystemInfoIndication
struct sNASSystemInfoIndication_MoreCDMA1xEVDOSystemInfo
```

```
{
    UINT16 mGeoSystemIndex;
};

// Structure to describe indication TLV 0x1C for NAS SystemInfoIndication
struct sNASSystemInfoIndication_MoreGSMSystemInfo
{
    UINT16 mGeoSystemIndex;
    eQMINASCellBroadcastCaps mCellBroadcastCapability;
};

// Structure to describe indication TLV 0x1D for NAS SystemInfoIndication
struct sNASSystemInfoIndication_MoreWCDMASystemInfo
{
    UINT16 mGeoSystemIndex;
    eQMINASCellBroadcastCaps mCellBroadcastCapability;
};

// Structure to describe indication TLV 0x1E for NAS SystemInfoIndication
struct sNASSystemInfoIndication_MoreLTESystemInfo
{
    UINT16 mGeoSystemIndex;
};

// Structure to describe indication TLV 0x1F for NAS SystemInfoIndication
struct sNASSystemInfoIndication_GSMCallBarring
{
    eQMINASCallBarringStatus mCSCallBarringStatus;
    eQMINASCallBarringStatus mPSCallBarringStatus;
};

// Structure to describe indication TLV 0x20 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_WCDMACallBarring
{
    eQMINASCallBarringStatus mCSCallBarringStatus;
    eQMINASCallBarringStatus mPSCallBarringStatus;
};
```

```
// Structure to describe indication TLV 0x21 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_LTEVoice
{
    INT8 mLTEVoiceSupported;
};

// Structure to describe indication TLV 0x22 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_GSMCipher
{
    eQMINASServiceDomains mCipheringOnServiceDomain;
};

// Structure to describe indication TLV 0x23 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_WCDMACipher
{
    eQMINASServiceDomains mCipheringOnServiceDomain;
};

// Structure to describe indication TLV 0x24 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_NoPLMNChange
{
    INT8 mNoPLMNChange;
};

// Structure to describe indication TLV 0x25 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_TDSCDMAServiceStatusInfo
{
    eQMINASServiceStatus mServiceStatus;
    eQMINASServiceStatus mTrueServiceStatus;
    eQMINASPreferredDataBath mPreferredDataPath;
};

// Structure to describe indication TLV 0x26 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_TDSCDMASystemInfo
{
    INT8 mServiceDomainValid;
```

```
    eQMINASSystemServiceCapabilities mServiceDomain;
    INT8 mServiceCapabilityValid;
    eQMINASSystemServiceCapabilities mSystemServiceCapabilities;
    INT8 mRoamStatusValid;
    eQMINASRoamStatus mRoamStatus;
    INT8 mSystemForbiddenValid;
    eQMINASSystemForbidden mSystemForbidden;
    INT8 mLocationAreaCodeValid;
    UINT16 mLocationAreaCode;
    INT8 mCellIDValid;
    UINT32 mCellID;
    INT8 mRegistrationRejectInformationValid;
    eQMINASSystemServiceCapabilities mRegistrationRejectServiceDomain;
    UINT8 mRejectCause;
    INT8 mNetworkIDValid;
    char mMobileCountryCode[3];
    char mMobileNetworkCode[3];
    INT8 mHighSpeedCallStatusValid;
    eQMINASHighSpeedCallStatus mHighSpeedCallStatus;
    INT8 mHighSpeedServiceIndicationValid;
    eQMINASHighSpeedCallStatus mHighSpeedServiceIndication;
    INT8 mCellParameterIDValid;
    UINT16 mCellParameterID;
    INT8 mCellBroadcastCapabilityValid;
    eQMINASCellBroadcastCaps2 mCellBroadcastCapability;
    INT8 mCSBarringStatusValid;
    eQMINASCallBarringStatus mCSCallBarringStatus;
    INT8 mPSBarringStatusValid;
    eQMINASCallBarringStatus mPSCallBarringStatus;
    INT8 mCipheringValid;
    eQMINASServiceDomains mCipheringOnServiceDomain;
};

// Structure to describe indication TLV 0x27 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_EMBMSCoverage
{
    INT8 mEMBMSSupported;
```

```
};

// Structure to describe indication TLV 0x28 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_SIMRejectInfo
{
    eQMINASSIMRejectStates mSIMRejectInfo;
};

// Structure to describe indication TLV 0x29 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_WCDMAEUTRADetection
{
    eQMINASEUTRAStatus mEUTRADetectionStatus;
};

// Structure to describe indication TLV 0x2A for NAS SystemInfoIndication
struct sNASSystemInfoIndication_LTEIMSVoice
{
    INT8 mIMSVoiceSupportAvailable;
};

// Structure to describe indication TLV 0x2B for NAS SystemInfoIndication
struct sNASSystemInfoIndication_LTEVoiceDomain
{
    eQMINASLTEVoiceDomains mLTEVoiceDomain;
};

// Structure to describe indication TLV 0x2C for NAS SystemInfoIndication
struct sNASSystemInfoIndication_CDMARegZoneID
{
    UINT16 mCDMARegZoneID;
};

// Structure to describe indication TLV 0x2D for NAS SystemInfoIndication
struct sNASSystemInfoIndication_GSMRAC
{
    UINT8 mGSMRoutingAreaCode;
};
```

```
// Structure to describe indication TLV 0x2E for NAS SystemInfoIndication
struct sNASSystemInfoIndication_WCDMARAC
{
    UINT8 mWCDMARoutingAreaCode;
};

// Structure to describe indication TLV 0x2F for NAS SystemInfoIndication
struct sNASSystemInfoIndication_CDMAResolvedMCC
{
    UINT16 mCDMAMCCResolvedViaSIDLookup;
};

// Structure to describe indication TLV 0x30 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_RegistrationRestriction
{
    eQMINASRegistrationRestrictions mRegistrationRestriction;
};

// Structure to describe indication TLV 0x31 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_TDSCDMARegistrationDomain
{
    eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
};

// Structure to describe indication TLV 0x32 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_LTERegistrationDomain
{
    eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
};

// Structure to describe indication TLV 0x33 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_WCDMARegistrationDomain
{
    eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
};
```

```
// Structure to describe indication TLV 0x34 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_GSMRegistrationDomain
{
    eQMINASRegistrationDomains mTDSCDMARegistrationDomain;
};

// Structure to describe indication TLV 0x35 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_EMBMSTraceID
{
    INT16 mEMBMSTraceID;
};

// Structure to describe indication TLV 0x36 for NAS SystemInfoIndication
struct sNASSystemInfoIndication_WCDMACSGInfo
{
    UINT32 mCSGID;
    UINT8 mNameLength;

    // This array must be the size specified by mNameLength
    // wchar_t mName[1];
};
```

Example:

```
int sysIndPro = 0;
void __cdecl NasSysInfoIdnCallback(
    ULONG                           svcID,
    ULONG                           msgID,
    GOBIHANDLE                       /* handle */,
    ULONG                           outLen,
    const BYTE *                    pOut )
{
    if (svcID != 3 || msgID != 78)
    {
        return;
    }
    if(sysIndPro)
    {
            return ;
    }

    sysIndPro = 1;
    ULONG state = ULONG_MAX;
    ULONG cer = ULONG_MAX;


    std::map <UINT8, const sQMIRawContentHeader *> tlvs = GetTLVs( &pOut[0], outLen );
```

```
    //if(mDataRadioVal == eQMINASRadioInterfaces_UMTS)//wcdma
    {
            //cdma
            std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x10 );
            if (pIter != tlvs.end())
            {
                    const sQMIRawContentHeader * pTmp = pIter->second;
                    if (pTmp->mLength >= sizeof (sNASSystemInfoIndication_CDMAServiceStatusInfo))
                    {
                            pTmp++;
                            const sNASSystemInfoIndication_CDMAServiceStatusInfo * pState =
                                    (const sNASSystemInfoIndication_CDMAServiceStatusInfo *)pTmp;

                           eQMINASServiceStatus x =  pState->mServiceStatus;
                    }
            }
            //std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x13
);
            //wcdma
            pIter = tlvs.find( 0x13 );
            if (pIter != tlvs.end())
            {
                    const sQMIRawContentHeader * pTmp = pIter->second;
                   if (pTmp->mLength >= sizeof (sNASSystemInfoIndication_WCDMAServiceStatusInfo))
                    {
                            pTmp++;
                            const sNASSystemInfoIndication_WCDMAServiceStatusInfo * pState =
                                    (const sNASSystemInfoIndication_WCDMAServiceStatusInfo *)pTmp;

                            g_CurrentNetworkType = NETWORK_MODE_WCDMA;
                            switch(pState->mServiceStatus)
                            {
                            case eQMINASServiceStatus_NoService:
                                    g_CurrentNetworkType = NETWORK_MODE_NULL;
                                    break;
                           case eQMINASServiceStatus_LimitedService:
                                    g_InternetType = Internet_Type_Umts;
                                    g_bStateGsmWcdma = STATE_NETWORK_MODE_WCDMA_ABN
ORMAL;
                                    break;
                           default:
                                    g_InternetType = Internet_Type_Umts;
                                    g_bStateGsmWcdma = STATE_NETWORK_MODE_WCDMA_NOR
MAL;
                                    break;
                            }
                    }
            }

            pIter = tlvs.find( 0x18 );
            if (pIter != tlvs.end())
            {
                    const sQMIRawContentHeader * pTmp = pIter->second;
                    if (pTmp->mLength >= sizeof (sNASSystemInfoIndication_WCDMASystemInfo))
                    {
```

```
                        pTmp++;
                        const sNASSystemInfoIndication_WCDMASystemInfo * pState =
                                (const sNASSystemInfoIndication_WCDMASystemInfo *)pTmp;

                        char MCC[3]={0},MNC[3]={0};
                        strncpy(MCC,pState->mMobileCountryCode,3);
                        strncpy(MNC,pState->mMobileNetworkCode,3);

                        //strcpy( m_plmnmode.plmn, s.LockBuffer() );
                       //UpdateSPName();
                    }
                }
    }

    sysIndPro = 0;
}


ULONG ParseGetSysInfo(
    ULONG                   inLen,
    const BYTE *          pIn,
    int * serviceStatus,
    int * networkType,
    UINT16 * pMcc,
    UINT16 * pMnc)
{
    // Validate arguments
    if (pIn == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

     // Find the TLV
    const sNASGetSystemInfoResponse_CDMAServiceStatusInfo * pTLVx10;
    ULONG outLenx10;
    ULONG rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
    if (rc == eGOBI_ERR_NONE)
    {
        //process cdma
            return eGOBI_ERR_NONE;
    }

    // Find the TLV
    const sNASGetSystemInfoResponse_GSMServiceStatusInfo * pTLVx12;
    ULONG outLenx12;
    rc = GetTLV( inLen, pIn, 0x12, &outLenx12, (const BYTE **)&pTLVx12 );
    if (rc == eGOBI_ERR_NONE)
    {
        //process gsm
            const sNASGetSystemInfoResponse_GSMSystemInfo * pTLVx17;
                ULONG outLenx17;
                rc = GetTLV( inLen, pIn, 0x17, &outLenx17, (const BYTE **)&pTLVx17 );

                if (rc == eGOBI_ERR_NONE)
                {
```

```
                        *networkType = eQMIWDSDataSystems_GPRS;
                        *serviceStatus = pTLVx12->mServiceStatus;
                        UINT16 mcc = 0;
                        UINT16 mnc = 0;
                        //mcc
                        if((pTLVx17->mMobileCountryCode[2] >= '0')&&(pTLVx17->mMobileCountryC
ode[2] <= '9'))
                        {
                                mcc += (pTLVx17->mMobileCountryCode[2]-0x30);
                                if((pTLVx17->mMobileCountryCode[1] >= '0')&&(pTLVx17->mMobile
CountryCode[1] <= '9'))
                                {
                                        mcc += (pTLVx17->mMobileCountryCode[1]-0x30)*10;
                                        if((pTLVx17->mMobileCountryCode[0] >= '0')&&(pTLVx17->
mMobileCountryCode[0] <= '9'))
                                        {
                                                mcc += (pTLVx17->mMobileCountryCode[0]-0x30)*
100;
                                        }
                                }
                        }
                        else if((pTLVx17->mMobileCountryCode[1] >= '0')&&(pTLVx17->mMobileCou
ntryCode[1] <= '9'))
                        {
                                mcc += (pTLVx17->mMobileCountryCode[1]-0x30);
                                if((pTLVx17->mMobileCountryCode[0] >= '0')&&(pTLVx17->mMobile
CountryCode[0] <= '9'))
                                {
                                        mcc += (pTLVx17->mMobileCountryCode[0]-0x30)*10;
                                }
                        }
                        else if((pTLVx17->mMobileCountryCode[0] >= '0')&&(pTLVx17->mMobileCou
ntryCode[0] <= '9'))
                        {
                                mcc += (pTLVx17->mMobileCountryCode[0]-0x30);
                        }

                        //mnc
                        if((pTLVx17->mMobileNetworkCode[2] >= '0')&&(pTLVx17->mMobileNetwork
Code[2] <= '9'))
                        {
                                mnc += (pTLVx17->mMobileNetworkCode[2]-0x30);
                                if((pTLVx17->mMobileNetworkCode[1] >= '0')&&(pTLVx17->mMobile
NetworkCode[1] <= '9'))
                                {
                                        mnc += (pTLVx17->mMobileNetworkCode[1]-0x30)*10;
                                        if((pTLVx17->mMobileNetworkCode[0] >= '0')&&(pTLVx17-
>mMobileNetworkCode[0] <= '9'))
                                        {
                                                mnc += (pTLVx17->mMobileNetworkCode[0]-0x30)
*100;
                                        }
                                }
                        }
                        else if((pTLVx17->mMobileNetworkCode[1] >= '0')&&(pTLVx17->mMobileNet
```

```
workCode[1] <= '9'))
                        {
                                mnc += (pTLVx17->mMobileNetworkCode[1]-0x30);
                                if((pTLVx17->mMobileNetworkCode[0] >= '0')&&(pTLVx17->mMobile
NetworkCode[0] <= '9'))
                                {
                                        mnc += (pTLVx17->mMobileNetworkCode[0]-0x30)*10;
                                }
                        }
                        else if((pTLVx17->mMobileNetworkCode[0] >= '0')&&(pTLVx17->mMobileNet
workCode[0] <= '9'))
                        {
                                mnc += (pTLVx17->mMobileNetworkCode[0]-0x30);
                        }
                        *pMcc = mcc;
                        *pMnc = mnc;
                        return eGOBI_ERR_NONE;
                }
    }

    // Find the TLV
    const sNASGetSystemInfoResponse_WCDMAServiceStatusInfo * pTLVx13;
    ULONG outLenx13;
    rc = GetTLV( inLen, pIn, 0x13, &outLenx13, (const BYTE **)&pTLVx13 );
    if (rc == eGOBI_ERR_NONE)
    {
        //process wcdma
            const sNASGetSystemInfoResponse_WCDMASystemInfo * pTLVx18;
            ULONG outLenx18;
            rc = GetTLV( inLen, pIn, 0x18, &outLenx18, (const BYTE **)&pTLVx18 );

            if (rc == eGOBI_ERR_NONE)
            {
                    *networkType = eQMIWDSDataSystems_WCDMA;
                    *serviceStatus = pTLVx13->mServiceStatus;
                    UINT16 mcc = 0;
                    UINT16 mnc = 0;
                    //mcc
                   if((pTLVx18->mMobileCountryCode[2] >= '0')&&(pTLVx18->mMobileCountryCode[2]
 <= '9'))
                    {
                            mcc += (pTLVx18->mMobileCountryCode[2]-0x30);
                            if((pTLVx18->mMobileCountryCode[1] >= '0')&&(pTLVx18->mMobileCount
ryCode[1] <= '9'))
                            {
                                    mcc += (pTLVx18->mMobileCountryCode[1]-0x30)*10;
                                    if((pTLVx18->mMobileCountryCode[0] >= '0')&&(pTLVx18->mMo
bileCountryCode[0] <= '9'))
                                    {
                                            mcc += (pTLVx18->mMobileCountryCode[0]-0x30)*100;
                                    }
                            }
                    }
                    else if((pTLVx18->mMobileCountryCode[1] >= '0')&&(pTLVx18->mMobileCountryC
ode[1] <= '9'))
```

```
                {
                        mcc += (pTLVx18->mMobileCountryCode[1]-0x30);
                        if((pTLVx18->mMobileCountryCode[0] >= '0')&&(pTLVx18->mMobile
CountryCode[0] <= '9'))
                        {
                                mcc += (pTLVx18->mMobileCountryCode[0]-0x30)*10;
                        }
                }
                else if((pTLVx18->mMobileCountryCode[0] >= '0')&&(pTLVx18->mMobileCountryC
ode[0] <= '9'))
                {
                        mcc += (pTLVx18->mMobileCountryCode[0]-0x30);
                }

                //mnc
                if((pTLVx18->mMobileNetworkCode[2] >= '0')&&(pTLVx18->mMobileNetworkCode
[2] <= '9'))
                {
                        mnc += (pTLVx18->mMobileNetworkCode[2]-0x30);
                        if((pTLVx18->mMobileNetworkCode[1] >= '0')&&(pTLVx18->mMobileNetw
orkCode[1] <= '9'))
                        {
                                mnc += (pTLVx18->mMobileNetworkCode[1]-0x30)*10;
                                if((pTLVx18->mMobileNetworkCode[0] >= '0')&&(pTLVx18->mMo
bileNetworkCode[0] <= '9'))
                                {
                                        mnc += (pTLVx18->mMobileNetworkCode[0]-0x30)*100;
                                }
                        }
                }
                else if((pTLVx18->mMobileNetworkCode[1] >= '0')&&(pTLVx18->mMobileNetwork
Code[1] <= '9'))
                {
                        mnc += (pTLVx18->mMobileNetworkCode[1]-0x30);
                        if((pTLVx18->mMobileNetworkCode[0] >= '0')&&(pTLVx18->mMobile
NetworkCode[0] <= '9'))
                        {
                                mnc += (pTLVx18->mMobileNetworkCode[0]-0x30)*10;
                        }
                }
                else if((pTLVx18->mMobileNetworkCode[0] >= '0')&&(pTLVx18->mMobileNetwork
Code[0] <= '9'))
                {
                        mnc += (pTLVx18->mMobileNetworkCode[0]-0x30);
                }
                *pMcc = mcc;
                *pMnc = mnc;

                return eGOBI_ERR_NONE;
        }
    }

    // Find the TLV
    const sNASGetSystemInfoResponse_LTEServiceStatusInfo * pTLVx14;
    ULONG outLenx14;
```

```
    rc = GetTLV( inLen, pIn, 0x14, &outLenx14, (const BYTE **)&pTLVx14 );
    if (rc == eGOBI_ERR_NONE)
    {
        //process lte
            const sNASGetSystemInfoResponse_LTESystemInfo * pTLVx19;
            ULONG outLenx19;
            rc = GetTLV( inLen, pIn, 0x19, &outLenx19, (const BYTE **)&pTLVx19 );

            if (rc == eGOBI_ERR_NONE)
            {
                    *networkType = eQMIWDSDataSystems_LTE;
                    *serviceStatus = pTLVx14->mServiceStatus;
                    UINT16 mcc = 0;
                    UINT16 mnc = 0;
                    //mcc
                   if((pTLVx19->mMobileCountryCode[2] >= '0')&&(pTLVx19->mMobileCountryCode[2]
 <= '9'))
                    {
                            mcc += (pTLVx19->mMobileCountryCode[2]-0x30);
                            if((pTLVx19->mMobileCountryCode[1] >= '0')&&(pTLVx19->mMobileCount
ryCode[1] <= '9'))
                            {
                                    mcc += (pTLVx19->mMobileCountryCode[1]-0x30)*10;
                                    if((pTLVx19->mMobileCountryCode[0] >= '0')&&(pTLVx19->mMo
bileCountryCode[0] <= '9'))
                                    {
                                            mcc += (pTLVx19->mMobileCountryCode[0]-0x30)*100;
                                    }
                            }
                    }
                    else if((pTLVx19->mMobileCountryCode[1] >= '0')&&(pTLVx19->mMobileCountryC
ode[1] <= '9'))
                    {
                            mcc += (pTLVx19->mMobileCountryCode[1]-0x30);
                            if((pTLVx19->mMobileCountryCode[0] >= '0')&&(pTLVx19->mMobile
CountryCode[0] <= '9'))
                            {
                                    mcc += (pTLVx19->mMobileCountryCode[0]-0x30)*10;
                            }
                    }
                    else if((pTLVx19->mMobileCountryCode[0] >= '0')&&(pTLVx19->mMobileCountryC
ode[0] <= '9'))
                    {
                            mcc += (pTLVx19->mMobileCountryCode[0]-0x30);
                    }

                    //mnc
                    if((pTLVx19->mMobileNetworkCode[2] >= '0')&&(pTLVx19->mMobileNetworkCode
[2] <= '9'))
                    {
                            mnc += (pTLVx19->mMobileNetworkCode[2]-0x30);
                            if((pTLVx19->mMobileNetworkCode[1] >= '0')&&(pTLVx19->mMobileNetw
orkCode[1] <= '9'))
                            {
                                    mnc += (pTLVx19->mMobileNetworkCode[1]-0x30)*10;
```

```
                                        if((pTLVx19->mMobileNetworkCode[0] >= '0')&&(pTLVx19->mMo
bileNetworkCode[0] <= '9'))
                                        {
                                                mnc += (pTLVx19->mMobileNetworkCode[0]-0x30)*100;
                                        }
                                }
                        }
                        else if((pTLVx19->mMobileNetworkCode[1] >= '0')&&(pTLVx19->mMobileNetwork
Code[1] <= '9'))
                        {
                                mnc += (pTLVx19->mMobileNetworkCode[1]-0x30);
                                if((pTLVx19->mMobileNetworkCode[0] >= '0')&&(pTLVx19->mMobile
NetworkCode[0] <= '9'))
                                {
                                        mnc += (pTLVx19->mMobileNetworkCode[0]-0x30)*10;
                                }
                        }
                        else if((pTLVx19->mMobileNetworkCode[0] >= '0')&&(pTLVx19->mMobileNetwork
Code[0] <= '9'))
                        {
                                mnc += (pTLVx19->mMobileNetworkCode[0]-0x30);
                        }
                        *pMcc = mcc;
                        *pMnc = mnc;
                        return eGOBI_ERR_NONE;
                }
        }


        return eGOBI_ERR_GENERAL;
}


NasSysIndCB(NasSysInfoIdnCallback);

        ULONG lo = 1024;
        BYTE rsp[1024] = { 0 };
        status = NASGetSystemInfo ( mhGobi, 2000, 0, 0, &lo, &rsp[0] );
        if (status != 0)
        {
            return status;
        }

        status = ParseGetSysInfo( lo,
                                        &rsp[0],
                                        &serviceStatus,
                                                                &networkType,
                                                                &mcc,&mnc);
```

3.1.22. NASSetRegistrationEventReport

Description	Sets the registration state for different QMI_NAS indications for the requesting control point
Request	**Optional TLVs**

Structure	// Structure to describe request TLV 0x10 for NASSetRegistrationEventReport()
	struct sNASSetRegistrationEventReportRequest_SystemSelectIndicator
	{
	INT8 mReportSystemSelect;
	};
	
	// Structure to describe request TLV 0x12 for NASSetRegistrationEventReport()
	struct sNASSetRegistrationEventReportRequest_DDTMIndicator
	{
	INT8 mReportDDTM;
	};
	
	// Structure to describe request TLV 0x13 for NASSetRegistrationEventReport()
	struct sNASSetRegistrationEventReportRequest_ServingSystemIndicator
	{
	INT8 mReportServingSystem;
	};
	
	// Structure to describe request TLV 0x14 for NASSetRegistrationEventReport()
	struct sNASSetRegistrationEventReportRequest_DualStandbyIndicator
	{
	INT8 mReportDualStandby;
	};
	
	// Structure to describe request TLV 0x15 for NASSetRegistrationEventReport()
	struct sNASSetRegistrationEventReportRequest_SubscriptionInformationIndicator
	{
	INT8 mReportSubscriptionInformation;
	};
	
	// Structure to describe request TLV 0x17 for NASSetRegistrationEventReport()
	struct sNASSetRegistrationEventReportRequest_NetworkTimeIndicator
	{
	INT8 mReportNetworkTime;
	};
	
	// Structure to describe request TLV 0x18 for NASSetRegistrationEventReport()

```
struct sNASSetRegistrationEventReportRequest_SystemInformationIndicator
{
    INT8 mReportSystemInformation;
};

// Structure to describe request TLV 0x19 for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_SignalStrengthIndicator
{
    INT8 mReportSignalStrength;
};

// Structure to describe request TLV 0x1A for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_ErrorRateIndicator
{
    INT8 mReportErrorRate;
};

// Structure to describe request TLV 0x1B for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_NewEVDOUATIIndicator
{
    INT8 mReportNewEVDOUATI;
};

// Structure to describe request TLV 0x1C for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_EVDOSessionIndicator
{
    INT8 mReportEVDOSessionClose;
};

// Structure to describe request TLV 0x1D for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_ManagedRoamingIndicator
{
    INT8 mReportManagedRoaming;
};

// Structure to describe request TLV 0x1E for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_CurrentPLMNName
```

```
{
    INT8 mReportCurrentPLMNName;
};

// Structure to describe request TLV 0x1F for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_EMBMSStatus
{
    INT8 mReportEMBMSStatus;
};

// Structure to describe request TLV 0x20 for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_RFBandInfo
{
    INT8 mReportRFBandInfo;
};

// Structure to describe request TLV 0x21 for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_NetworkReject
{
    INT8 mNetworkRejectEnabled;
    INT8 mSupressSytemInfoEnabled;
};

// Structure to describe request TLV 0x22 for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_OperatorNameData
{
    INT8 mOperatorNameDataEnabled;
};

// Structure to describe request TLV 0x23 for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_CSPPLMNModeBit
{
    INT8 mCSPPLMNModeBitEnabled;
};

// Structure to describe request TLV 0x24 for NASSetRegistrationEventReport()
struct sNASSetRegistrationEventReportRequest_RTREConfiguration
```

	{
	INT8 mRTREConfigurationEnabled;
	};

Response Structure	NULL
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

```
ULONG PackEventReportReg(
    ULONG *                 pOutLen,
    BYTE *                  pOut )
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx10Sz = sizeof( sNASSetRegistrationEventReportRequest_SystemSelectIndicator );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx10Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x10;
    pHeader->mLength = tlvx10Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sNASSetRegistrationEventReportRequest_SystemSelectIndicator * pTLVx10;
    pTLVx10 = (sNASSetRegistrationEventReportRequest_SystemSelectIndicator*)(pOut + offset);
    memset( pTLVx10, 0, tlvx10Sz );

    // Set the values
    pTLVx10->mReportSystemSelect = 1;

    offset += tlvx10Sz;


    WORD tlvx13Sz = sizeof( sNASSetRegistrationEventReportRequest_ServingSystemIndicator );
     pHeader = (sQMIRawContentHeader*)(pOut + offset);
          pHeader->mTypeID = 0x13;
          pHeader->mLength = tlvx13Sz;
```

```
        offset += sizeof( sQMIRawContentHeader );

        sNASSetRegistrationEventReportRequest_ServingSystemIndicator * pTLVx13;
    pTLVx13 = (sNASSetRegistrationEventReportRequest_ServingSystemIndicator*)(pOut + offset);
    memset( pTLVx13, 0, tlvx13Sz );

    // Set the values
    pTLVx13->mReportServingSystem = 0;

    offset += tlvx13Sz;


    WORD tlvx18Sz = sizeof( sNASSetRegistrationEventReportRequest_SystemInformationIndicator );
     pHeader = (sQMIRawContentHeader*)(pOut + offset);
        pHeader->mTypeID = 0x18;
        pHeader->mLength = tlvx18Sz;

        offset += sizeof( sQMIRawContentHeader );

        sNASSetRegistrationEventReportRequest_SystemInformationIndicator * pTLVx18;
    pTLVx18 = (sNASSetRegistrationEventReportRequest_SystemInformationIndicator*)(pOut + offset);
    memset( pTLVx18, 0, tlvx18Sz );

    // Set the values
    pTLVx18->mReportSystemInformation = 1;

    offset += tlvx18Sz;


    WORD tlvx19Sz = sizeof( sNASSetRegistrationEventReportRequest_SignalStrengthIndicator );
     pHeader = (sQMIRawContentHeader*)(pOut + offset);
        pHeader->mTypeID = 0x19;
        pHeader->mLength = tlvx19Sz;

        offset += sizeof( sQMIRawContentHeader );

        sNASSetRegistrationEventReportRequest_SignalStrengthIndicator * pTLVx19;
    pTLVx19 = (sNASSetRegistrationEventReportRequest_SignalStrengthIndicator*)(pOut + offset);
    memset( pTLVx19, 0, tlvx19Sz );

    // Set the values
    pTLVx19->mReportSignalStrength = 1;

    offset += tlvx19Sz;


    *pOutLen = offset;

    return eGOBI_ERR_NONE;
}


    ULONG li = 1024;
    BYTE req[1024] = { 0 };
```

```
PackEventReportReg(&li,&req[0]);
status = NASSetRegistrationEventReport ( mhGobi, 2000, li, &req[0],0,0 );
```

3.1.23. NASGetSignalInfo

Description	Queries information regarding the signal strength
Request Structure	NULL

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for NASGetSignalInfo()
	struct sNASGetSignalInfoResponse_CDMASignalInfo
	{
	INT8 mRSSI;
	UINT16 mECIO;
	};
	
	// Structure to describe response TLV 0x11 for NASGetSignalInfo()
	struct sNASGetSignalInfoResponse_CDMA1xEVDOSignalInfo
	{
	INT8 mRSSI;
	UINT16 mECIO;
	eQMINASSINRLevels mSINR;
	UINT32 mIO;
	};
	
	// Structure to describe response TLV 0x12 for NASGetSignalInfo()
	struct sNASGetSignalInfoResponse_GSMSignalInfo
	{
	INT8 mRSSI;
	};
	
	// Structure to describe response TLV 0x13 for NASGetSignalInfo()
	struct sNASGetSignalInfoResponse_WCDMASignalInfo
	{
	INT8 mRSSI;
	UINT16 mECIO;
	};

	// Structure to describe response TLV 0x14 for NASGetSignalInfo()
	struct sNASGetSignalInfoResponse_LTESignalInfo
	{
	INT8 mRSSI;
	INT8 mRSRQ;
	INT16 mRSRP;
	INT16 mSNR;
	};
	
	// Structure to describe response TLV 0x15 for NASGetSignalInfo()
	struct sNASGetSignalInfoResponse_TDSCDMASignalInfo
	{
	INT8 mPCCPCHRSCP;
	};
	
	// Structure to describe response TLV 0x16 for NASGetSignalInfo()
	struct sNASGetSignalInfoResponse_TDSCDMASignalInfoEx
	{
	float mRSSIdBm;
	float mRSCPdBm;
	float mECIOdB;
	float mSINRdB;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	A sig_strength value of -125 dBm or lower is used to indicate No Signal.
MSG ID	QMI_NAS_CONFIG_SIG_INFO 0x0050

IND Structure	// Structure to describe indication TLV 0x10 for NAS SignalInfoIndication
	struct sNASSignalInfoIndication_CDMASignalInfo
	{
	INT8 mRSSI;
	UINT16 mECIO;
	};

```
// Structure to describe indication TLV 0x11 for NAS SignalInfoIndication
struct sNASSignalInfoIndication_CDMA1xEVDOSignalInfo
{
    INT8 mRSSI;
    UINT16 mECIO;
    eQMINASSINRLevels mSINR;
    UINT32 mIO;
};

// Structure to describe indication TLV 0x12 for NAS SignalInfoIndication
struct sNASSignalInfoIndication_GSMSignalInfo
{
    INT8 mRSSI;
};

// Structure to describe indication TLV 0x13 for NAS SignalInfoIndication
struct sNASSignalInfoIndication_WCDMASignalInfo
{
    INT8 mRSSI;
    UINT16 mECIO;
};

// Structure to describe indication TLV 0x14 for NAS SignalInfoIndication
struct sNASSignalInfoIndication_LTESignalInfo
{
    INT8 mRSSI;
    INT8 mRSRQ;
    INT16 mRSRP;
    INT16 mSNR;
};

// Structure to describe indication TLV 0x15 for NAS SignalInfoIndication
struct sNASSignalInfoIndication_TDSCDMASignalInfo
{
    INT8 mPCCPCHRSCP;
};
```

```
// Structure to describe indication TLV 0x16 for NAS SignalInfoIndication
struct sNASSignalInfoIndication_TDSCDMASignalInfoEx
{
    float mRSSIdBm;
    float mRSCPdBm;
    float mECIOdB;
    float mSINRdB;
};
```

Example:

```
ULONG cGobiCMDLL::NasSigIndCB( tFNGenericCallback pCallback )
{
    ULONG status = 1;
    if( mpFnSetGenericCallback == 0 || mhGobi == 0)
    {
        return status;
    }

    // Configure the callback with the API
    status = mpFnSetGenericCallback( mhGobi, 3, 81, pCallback );
    return status;
}

void __cdecl NasSigInfoIdnCallback(
    ULONG                           svcID,
    ULONG                           msgID,
    GOBIHANDLE                       /* handle */,
    ULONG                           outLen,
    const BYTE *                    pOut )
{
    if (svcID != 3 || msgID != 81)
    {
        return;
    }
    if(sigIndPro)
    {
            return ;
    }

    sigIndPro = 1;
    ULONG state = ULONG_MAX;
    ULONG cer = ULONG_MAX;


    std::map <UINT8, const sQMIRawContentHeader *> tlvs = GetTLVs( &pOut[0], outLen );
    //if(mDataRadioVal == eQMINASRadioInterfaces_UMTS)//wcdma
    {
            std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x10 );
            if (pIter != tlvs.end())
            {
```

```
                const sQMIRawContentHeader * pTmp = pIter->second;
                if (pTmp->mLength >= sizeof (sNASSignalInfoIndication_CDMASignalInfo))
                {
                        pTmp++;
                        const sNASSignalInfoIndication_CDMASignalInfo * pState =
                                (const sNASSignalInfoIndication_CDMASignalInfo *)pTmp;

                        pTheDialog->PostMessage(WM_SIG_UPDATE_IND, (WPARAM)pState->mRS
SI, 0);
                }
        }
        //std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x13
 );
        pIter = tlvs.find( 0x12 );
        if (pIter != tlvs.end())
        {
                const sQMIRawContentHeader * pTmp = pIter->second;
                if (pTmp->mLength >= sizeof (sNASSignalInfoIndication_GSMSignalInfo))
                {
                        pTmp++;
                        const sNASSignalInfoIndication_GSMSignalInfo * pState =
                                (const sNASSignalInfoIndication_GSMSignalInfo *)pTmp;

                        pTheDialog->PostMessage(WM_SIG_UPDATE_IND, (WPARAM)pState->mRS
SI, 0);
                }
        }

        pIter = tlvs.find( 0x13 );
        if (pIter != tlvs.end())
        {
                const sQMIRawContentHeader * pTmp = pIter->second;
                if (pTmp->mLength >= sizeof (sNASSignalInfoIndication_WCDMASignalInfo))
                {
                        pTmp++;
                        const sNASSignalInfoIndication_WCDMASignalInfo * pState =
                                (const sNASSignalInfoIndication_WCDMASignalInfo *)pTmp;

                        pTheDialog->PostMessage(WM_SIG_UPDATE_IND, (WPARAM)pState->mRS
SI, 0);
                }
        }

        pIter = tlvs.find( 0x14 );
        if (pIter != tlvs.end())
        {
                const sQMIRawContentHeader * pTmp = pIter->second;
                if (pTmp->mLength >= sizeof (sNASSignalInfoIndication_LTESignalInfo))
                {
                        pTmp++;
                        const sNASSignalInfoIndication_LTESignalInfo * pState =
                                (const sNASSignalInfoIndication_LTESignalInfo *)pTmp;

                        pTheDialog->PostMessage(WM_SIG_UPDATE_IND, (WPARAM)pState->mRS
SI, 0);
```

```
                    }
                }
    }

    sigIndPro = 0;
}

ULONG ParseGetSigInfo(
    ULONG                   inLen,
    const BYTE *            pIn,
    INT8 *                  pSignalStrength)
{
    // Validate arguments
    if (pSignalStrength == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    //cdma
    const sNASGetSignalInfoResponse_CDMASignalInfo * pTLVx10;
    ULONG outLenx10;
    ULONG rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
    if (rc == eGOBI_ERR_NONE)
    {
            // Is the TLV large enough?
            if (outLenx10 < sizeof( sNASGetSignalInfoResponse_CDMASignalInfo ))
            {
                    return eGOBI_ERR_MALFORMED_RSP;
            }

            *pSignalStrength = pTLVx10->mRSSI;
        return eGOBI_ERR_NONE;
    }

    //gsm
    const sNASGetSignalInfoResponse_GSMSignalInfo * pTLVx12;
    ULONG outLenx12;
    rc = GetTLV( inLen, pIn, 0x12, &outLenx12, (const BYTE **)&pTLVx12 );
    if (rc == eGOBI_ERR_NONE)
    {
            // Is the TLV large enough?
            if (outLenx12 < sizeof( sNASGetSignalInfoResponse_GSMSignalInfo ))
            {
                    return eGOBI_ERR_MALFORMED_RSP;
            }

            *pSignalStrength = pTLVx12->mRSSI;
         return eGOBI_ERR_NONE;
    }

    //wcdma
    const sNASGetSignalInfoResponse_WCDMASignalInfo * pTLVx13;
    ULONG outLenx13;
    rc = GetTLV( inLen, pIn, 0x13, &outLenx13, (const BYTE **)&pTLVx13 );
    if (rc == eGOBI_ERR_NONE)
```

```
    {
            // Is the TLV large enough?
            if (outLenx13 < sizeof( sNASGetSignalInfoResponse_WCDMASignalInfo ))
            {
                    return eGOBI_ERR_MALFORMED_RSP;
            }

            *pSignalStrength = pTLVx13->mRSSI;
        return eGOBI_ERR_NONE;
    }

    //lte
    const sNASGetSignalInfoResponse_LTESignalInfo * pTLVx14;
    ULONG outLenx14;
    rc = GetTLV( inLen, pIn, 0x14, &outLenx14, (const BYTE **)&pTLVx14 );
    if (rc == eGOBI_ERR_NONE)
    {
            // Is the TLV large enough?
            if (outLenx14 < sizeof( sNASGetSignalInfoResponse_LTESignalInfo ))
            {
                    return eGOBI_ERR_MALFORMED_RSP;
            }

            *pSignalStrength = pTLVx14->mRSSI;
        return eGOBI_ERR_NONE;
    }




    *pSignalStrength = SCHAR_MIN;

    return eGOBI_ERR_NO_SIGNAL;
}


    NasSigIndCB(NasSigInfoIdnCallback);
    ULONG lo = 1024;
    BYTE rsp[1024] = { 0 };
    status = NASGetSignalInfo ( mhGobi, 2000, 0, 0, &lo, &rsp[0] );
    if (status != 0)
    {
        return status;
    }

    INT8 rssi;
    if(eGOBI_ERR_NONE == ParseGetSigInfo(lo,&rsp[0],&rssi))
    {
            *pRssi = rssi;
    }
```

3.1.24. NASGetPLMNName

Description	Queries the operator name for a specified network

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for NASGetPLMNName()
	struct sNASGetPLMNNameRequest_PLMN
	{
	UINT16 mMobileCountryCode;
	UINT16 mMobileNetworkCode;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for NASGetPLMNName()
	struct sNASGetPLMNNameRequest_SupressSIMError
	{
	INT8 mSIMInitNotChecked;
	};
	
	// Structure to describe request TLV 0x11 for NASGetPLMNName()
	struct sNASGetPLMNNameRequest_MNCPCSDigitIncludeStatus
	{
	INT8 mMNCIncludesPCSDigit;
	};
	
	// Structure to describe request TLV 0x12 for NASGetPLMNName()
	struct sNASGetPLMNNameRequest_AlwaysSendPLMNName
	{
	INT8 mAlwaysSendPLMNName;
	};
	
	// Structure to describe request TLV 0x13 for NASGetPLMNName()
	struct sNASGetPLMNNameRequest_UseStaticTableOnly
	{
	INT8 mUseStaticTableOnly;
	};
	
	// Structure to describe request TLV 0x14 for NASGetPLMNName()
	struct sNASGetPLMNNameRequest_CSGID

	{
	UINT32 mCSGID;
	};
	
	// Structure to describe request TLV 0x15 for NASGetPLMNName()
	struct sNASGetPLMNNameRequest_RAT
	{
	eQMINASRadioAccessTechnologies mRadioAccessTechnology;
	};

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for NASGetPLMNName()
	struct sNASGetPLMNNameResponse_Name1
	{
	eQMINASPLMNNameEncodingSchemes mSPNEncoding;
	UINT8 mSPNLength;
	
	// This array must be the size specified by mSPNLength
	// UINT8 mSPN[1];
	};
	
	struct sNASGetPLMNNameResponse_Name2
	{
	eQMINASPLMNNameEncodingSchemes mPLMNShortEncoding;
	eQMINASPLMNNameCountryInitials mPLMNShortCountryInitials;
	eQMINASPLMNNameSpareBits mPLMNSpareBits;
	UINT8 mPLMNShortLength;
	
	// This array must be the size specified by mPLMNShortLength
	// UINT8 mPLMNShort[1];
	};
	
	struct sNASGetPLMNNameResponse_Name3
	{
	eQMINASPLMNNameEncodingSchemes mPLMNLongEncoding;
	eQMINASPLMNNameCountryInitials mPLMNLongCountryInitials;
	eQMINASPLMNNameSpareBits mPLMNLongBits;
	UINT8 mPLMNLongLength;

	// This array must be the size specified by mPLMNLongLength
	// UINT8 mPLMNLong[1];
	};
	
	struct sNASGetPLMNNameResponse_Name
	{
	sNASGetPLMNNameResponse_Name1 mNASGetPLMNNameResponse_Name1;
	sNASGetPLMNNameResponse_Name2 mNASGetPLMNNameResponse_Name2;
	sNASGetPLMNNameResponse_Name3 mNASGetPLMNNameResponse_Name3;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_INVALID_ARG Value field of one or more TLVs in the request message contains an invalid value QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_UIM_NOT_INITIALIZED UIM is not initialized QMI_ERR_OP_DEVICE_ UNSUPPORTED Operation is not supported by the device
Constraint	

Example:

```
ULONG PackGetPLMNName(
    ULONG *              pOutLen,
    BYTE *               pOut,
    USHORT               mcc,
    USHORT               mnc )
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Check size
    WORD tlvx01Sz = sizeof( sNASGetPLMNNameRequest_PLMN );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }
```

```
    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sNASGetPLMNNameRequest_PLMN * pTLVx01;
    pTLVx01 = (sNASGetPLMNNameRequest_PLMN*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the values
    pTLVx01->mMobileCountryCode = mcc;
    pTLVx01->mMobileNetworkCode = mnc;

    offset += tlvx01Sz;

    WORD tlvx12Sz = sizeof( sNASGetPLMNNameRequest_AlwaysSendPLMNName );
    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x12;
    pHeader->mLength = tlvx12Sz;

    offset += sizeof( sQMIRawContentHeader );

    sNASGetPLMNNameRequest_AlwaysSendPLMNName * pTLVx12;
    pTLVx12 = (sNASGetPLMNNameRequest_AlwaysSendPLMNName*)(pOut + offset);
    memset( pTLVx12, 0, tlvx12Sz );

    pTLVx12->mAlwaysSendPLMNName = 1;
    offset += tlvx12Sz;

    *pOutLen = offset;

    return eGOBI_ERR_NONE;
}


ULONG ParseGetPLMNName(
    ULONG                 inLen,
    const BYTE *          pIn,
    ULONG *               pNamesSize,
    BYTE *                pNames )
{

    // Validate arguments
    if (pIn == 0 || *pNamesSize == 0 || pNames == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    const sNASGetPLMNNameResponse_Name * pTLVx10;
    ULONG outLenx10;
    ULONG rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
```

```
    }

    // The output format just happens to be the same as
    // sNASGetPLMNNameResponse_Name.  Copy the full TLV to pNames
    /*if (outLenx10 > *pNamesSize)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }*/
    BYTE tempName[100]={0};

    if(pTLVx10->mNASGetPLMNNameResponse_Name1.mSPNLength != 0)
    {
            BYTE * pName = (BYTE *)(pTLVx10)+sizeof(pTLVx10->mNASGetPLMNNameResponse_N
ame1);
            if(eQMINASPLMNNameEncodingSchemes_ASCII == pTLVx10->mNASGetPLMNNameRespo
nse_Name1.mSPNEncoding)
            {
                    memcpy( pNames,pName ,pTLVx10->mNASGetPLMNNameResponse_Name1.mSPN
Length);
                    *pNamesSize = pTLVx10->mNASGetPLMNNameResponse_Name1.mSPNLength;
            }
            else
            {
                    int size = WideCharToMultiByte(CP_ACP, 0,(LPCWCH)pName,pTLVx10->mNASGet
PLMNNameResponse_Name1.mSPNLength,(LPSTR)pNames, *pNamesSize, NULL, NULL);
                    *pNamesSize = size;
            }
    }
    else if(pTLVx10->mNASGetPLMNNameResponse_Name2.mPLMNShortLength != 0)
    {
            BYTE * pName = (BYTE *)(pTLVx10)+sizeof(pTLVx10->mNASGetPLMNNameResponse_N
ame1)+sizeof(pTLVx10->mNASGetPLMNNameResponse_Name2);
            if(eQMINASPLMNNameEncodingSchemes_ASCII == pTLVx10->mNASGetPLMNNameRespo
nse_Name2.mPLMNShortEncoding)
            {
                    memcpy( pNames,pName ,pTLVx10->mNASGetPLMNNameResponse_Name2.mPLM
NShortLength);
                    *pNamesSize = pTLVx10->mNASGetPLMNNameResponse_Name2.mPLMNShortLen
gth;
            }
            else
            {
                    int size = WideCharToMultiByte(CP_ACP, 0,(LPCWCH)pName,pTLVx10->mNASGet
PLMNNameResponse_Name2.mPLMNShortLength,(LPSTR)pNames, *pNamesSize, NULL, NULL);
                    *pNamesSize = size;
            }
    }


    return eGOBI_ERR_NONE;
}

    ULONG lo = 1024;
    BYTE rsp[1024] = { 0 };
    ULONG li = 1024;
```

```
BYTE req[1024] = { 0 };

PackGetPLMNName(&li,req,mcc,mnc);
status = mpFnNasGetPlmnName( mhGobi, 2000, li,  &req[0], &lo, &rsp[0] );
if (status != 0)
{
    return status;
}


status = ParseGetPLMNName( lo,&rsp[0],(ULONG *)&plmanNameLen,(BYTE *)plmnName);
```

3.1.25. UIMVerifyPIN

Description	Verifies the PIN before the card content is accessed

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for UIMVerifyPIN()
	struct sUIMVerifyPINRequest_SessionInfo
	{
	eQMIUIMSessionTypes mSessionType;
	UINT8 mAIDLength;
	
	// This array must be the size specified by mAIDLength
	// UINT8 mAID[1];
	};
	
	// Structure to describe request TLV 0x02 for UIMVerifyPIN()
	struct sUIMVerifyPINRequest_Verify
	{
	eQMIUIMPINIDs mID;
	UINT8 mPINLength;
	
	// This array must be the size specified by mPINLength
	// char mPINValue[1];
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for UIMVerifyPIN()
	struct sUIMVerifyPINRequest_EncryptedPIN1
	{
	UINT8 mPINLength;

	// This array must be the size specified by mPINLength
	// UINT8 mValue[1];
	};
	
	// Structure to describe request TLV 0x11 for UIMVerifyPIN()
	struct sUIMVerifyPINRequest_KeyReferenceID
	{
	eQMIUIMKeyReferenceID mKeyReferenceID;
	};
	
	// Structure to describe request TLV 0x12 for UIMVerifyPIN()
	struct sUIMVerifyPINRequest_ResponseInIndication
	{
	UINT32 mIndicationToken;
	};

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for UIMVerifyPIN()
	struct sUIMVerifyPINResponse_Retries
	{
	UINT8 mRemainingVerifyRetries;
	UINT8 mRemainingUnblockRetries;
	};
	
	// Structure to describe response TLV 0x11 for UIMVerifyPIN()
	struct sUIMVerifyPINResponse_EncryptedPIN1
	{
	UINT8 mPINLength;
	
	// This array must be the size specified by mPINLength
	// UINT8 mValue[1];
	};
	
	// Structure to describe response TLV 0x12 for UIMVerifyPIN()
	struct sUIMVerifyPINResponse_ResponseInIndication
	{

	UINT32 mIndicationToken; };
Return Value	QMI_ERR_NONE No error in the request. QMI_ERR_INTERNAL Unexpected error occurred during processing. QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission. QMI_ERR_NO_MEMORY Device could not allocate memory to formulate the response. QMI_ERR_NO_EFFECT Operation had no effect. QMI_ERR_ARG_TOO_LONG Device cannot handle the PIN length in the request. QMI_ERR_INCORRECT_PIN PIN in the request is incorrect. QMI_ERR_PIN_BLOCKED PIN is blocked. An unblock operation must be issued. QMI_ERR_PIN_PERM_BLOCKED PIN is permanently blocked. The SIM is unusable. QMI_ERR_SIM_NOT_INITIALIZED PIN is not yet initialized because the SIM initialization has not finished. Try the PIN operation later. QMI_ERR_OP_DEVICE_ UNSUPPORTED Operation is not supported by the device nor by the SIM card. QMI_ERR_MISSING_ARG TLV was missing in the request. QMI_ERR_INVALID_PINID PIN in the request is invalid. QMI_ERR_DEVICE_NOT_READY Device is not ready
Constraint	
MSG ID	QMI_UIM_VERIFY_PIN_IND 0x0026

IND Structure	// Structure to describe indication TLV 0x01 for UIM VerifyPINIndication
	struct sUIMVerifyPINIndication_OriginalToken
	{
	UINT32 mIndicationToken;
	};
	
	// Structure to describe indication TLV 0x10 for UIM VerifyPINIndication
	struct sUIMVerifyPINIndication_Retries
	{
	UINT8 mRemainingVerifyRetries;
	UINT8 mRemainingUnblockRetries;
	};

```
// Structure to describe indication TLV 0x11 for UIM VerifyPINIndication
struct sUIMVerifyPINIndication_EncryptedPIN1
{
    UINT8 mPINLength;

    // This array must be the size specified by mPINLength
    // UINT8 mValue[1];
};
```

Example:

```
ULONG PackUIMVerifyPIN(
    ULONG *                         pOutLen,
    BYTE *                          pOut,
    ULONG                           id,
    CHAR *                          pValue )
{
    // Validate arguments
    if (pOut == 0
    || id < 1
    || id > 2
    || pValue == 0
    || pValue[0] == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }


    WORD tlvx01Sz = sizeof( sUIMVerifyPINRequest_SessionInfo );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
    pHeader->mLength = tlvx01Sz;

    ULONG offset = sizeof( sQMIRawContentHeader );

    sUIMVerifyPINRequest_SessionInfo * pTLVx01;
    pTLVx01 = (sUIMVerifyPINRequest_SessionInfo*)(pOut + offset);
    memset( pTLVx01, 0, tlvx01Sz );

    // Set the values
    if(1)//(IsCdmaTechnology())//TODO:limingjun
    {
            pTLVx01->mSessionType = eQMIUIMSessionTypes_Primary1XProvisioning;
    }
    else
    {
            pTLVx01->mSessionType = eQMIUIMSessionTypes_PrimaryGWProvisioning;
```

```
    }

    pTLVx01->mAIDLength = 0;
    offset += sizeof( sUIMVerifyPINRequest_SessionInfo );

    // Add mPINValue
    //memcpy( (pOut + offset), val.c_str(), valSz );
    //offset += valSz;

    // Add arguments
    std::string val( pValue );
    UINT8 valSz = (UINT8)val.size();
    // Check size
    WORD tlvx02Sz = sizeof( sUIMVerifyPINRequest_Verify ) + valSz;
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx02Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut+offset);
    pHeader->mTypeID = 0x02;
    pHeader->mLength = tlvx02Sz;

    offset += sizeof( sQMIRawContentHeader );

    sUIMVerifyPINRequest_Verify * pTLVx02;
    pTLVx02 = (sUIMVerifyPINRequest_Verify*)(pOut + offset);
    memset( pTLVx02, 0, tlvx02Sz );

    // Set the values
    pTLVx02->mID = (eQMIUIMPINIDs)id;
    pTLVx02->mPINLength = valSz;
    offset += sizeof( sUIMVerifyPINRequest_Verify );

    // Add mPINValue
    memcpy( (pOut + offset), val.c_str(), valSz );
    offset += valSz;

    *pOutLen = offset;
    return eGOBI_ERR_NONE;
}

ULONG cGobiCMDLL::SetUimVerifyPINCB( tFNGenericCallback pCallback )
{
    ULONG status = 1;
    if( mpFnSetGenericCallback == 0 || mhGobi == 0)
    {
        return status;
    }

    // Configure the callback with the API
    status = mpFnSetGenericCallback( mhGobi, 11, 38, pCallback );
    return status;
}
```

```
SetUimVerifyPINCB(UimVerifyPINCallback);
ULONG li = 1024;
BYTE req[1024] = { 0 };
ULONG lo = 2048;
BYTE rsp[2048] = { 0 };

PackUIMVerifyPIN(&li,req,pINID,pIN);
status = mpFnUimVerifyPIN( mhGobi, 2000, li, &req[0],&lo,&rsp[0]);
```

3.1.26. UIMUnblockPIN

Description	Unblocks a blocked PIN using the PUK code

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for UIMUnblockPIN()
	struct sUIMUnblockPINRequest_SessionInfo
	{
	eQMIUIMSessionTypes mSessionType;
	UINT8 mAIDLength;
	
	// This array must be the size specified by mAIDLength
	// UINT8 mAID[1];
	};
	
	// Structure to describe request TLV 0x02 for UIMUnblockPIN()
	struct sUIMUnblockPINRequest_Unblock1
	{
	eQMIUIMPINIDs mID;
	UINT8 mPUKLength;
	
	// This array must be the size specified by mPUKLength
	// char mPUKValue[1];
	};
	
	struct sUIMUnblockPINRequest_Unblock2
	{
	UINT8 mNewPINLength;
	
	// This array must be the size specified by mNewPINLength
	// char mNewPINValue[1];
	};

	struct sUIMUnblockPINRequest_Unblock
	{
	sUIMUnblockPINRequest_Unblock1 mUIMUnblockPINRequest_Unblock1;
	sUIMUnblockPINRequest_Unblock2 mUIMUnblockPINRequest_Unblock2;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for UIMUnblockPIN()
	struct sUIMUnblockPINRequest_KeyReferenceID
	{
	eQMIUIMKeyReferenceID mKeyReferenceID;
	};
	
	// Structure to describe request TLV 0x11 for UIMUnblockPIN()
	struct sUIMUnblockPINRequest_ResponseInIndication
	{
	UINT32 mIndicationToken;
	};

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for UIMUnblockPIN()
	struct sUIMUnblockPINResponse_Retries
	{
	UINT8 mRemainingVerifyRetries;
	UINT8 mRemainingUnblockRetries;
	};
	
	// Structure to describe response TLV 0x11 for UIMUnblockPIN()
	struct sUIMUnblockPINResponse_EncryptedPIN1
	{
	UINT8 mPINLength;
	
	// This array must be the size specified by mPINLength
	// UINT8 mValue[1];
	};
	
	// Structure to describe response TLV 0x12 for UIMUnblockPIN()
	struct sUIMUnblockPINResponse_ResponseInIndication

	{
	UINT32 mIndicationToken;
	};

Return Value	QMI_ERR_NONE No error in the request. QMI_ERR_INTERNAL Unexpected error occurred during processing. QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission. QMI_ERR_NO_MEMORY Device could not allocate memory to formulate the response. QMI_ERR_NO_EFFECT Operation had no effect. QMI_ERR_ARG_TOO_LONG Device cannot handle the PIN length in the request. QMI_ERR_INCORRECT_PIN PIN in the request is incorrect. QMI_ERR_PIN_BLOCKED PIN is blocked. An unblock operation must be issued. QMI_ERR_PIN_PERM_BLOCKED PIN is permanently blocked. The SIM is unusable. QMI_ERR_SIM_NOT_INITIALIZED PIN is not yet initialized because the SIM initialization has not finished. Try the PIN operation later. QMI_ERR_OP_DEVICE_ UNSUPPORTED Operation is not supported by the device nor by the SIM card. QMI_ERR_MISSING_ARG TLV was missing in the request. QMI_ERR_INVALID_PINID PIN in the request is invalid. QMI_ERR_DEVICE_NOT_READY Device is not ready.
Constraint	
MSG ID	QMI_UIM_UNBLOCK_PIN_IND 0x0027

IND Structure	// Structure to describe indication TLV 0x01 for UIM UnblockPINIndication
	struct sUIMUnblockPINIndication_OriginalToken
	{
	UINT32 mIndicationToken;
	};
	
	// Structure to describe indication TLV 0x10 for UIM UnblockPINIndication
	struct sUIMUnblockPINIndication_Retries
	{
	UINT8 mRemainingVerifyRetries;
	UINT8 mRemainingUnblockRetries;
	};

```
// Structure to describe indication TLV 0x11 for UIM UnblockPINIndication
struct sUIMUnblockPINIndication_EncryptedPIN1
{
    UINT8 mPINLength;

    // This array must be the size specified by mPINLength
    // UINT8 mValue[1];
};
```

Example:

```
ULONG cGobiCMDLL::SetUimUnblockPINCB( tFNGenericCallback pCallback )
{
    ULONG status = 1;
    if( mpFnSetGenericCallback == 0 || mhGobi == 0)
    {
        return status;
    }

    // Configure the callback with the API
    status = mpFnSetGenericCallback( mhGobi, 11, 39, pCallback );
    return status;
}

ULONG PackUIMUnblockPIN(
    ULONG *                         pOutLen,
    BYTE *                         pOut,
    ULONG                           id,
    CHAR *                         pPUKValue,
    CHAR *                         pNewValue )
{
    // Validate arguments
    if (pOut == 0
    ||  id < 1
    ||  id > 2
    ||  pPUKValue == 0
    ||  pPUKValue[0] == 0
    ||  pNewValue == 0
    ||  pNewValue[0] == 0 )
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    WORD tlvx01Sz = sizeof( sUIMUnblockPINRequest_SessionInfo );
    if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx01Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    sQMIRawContentHeader * pHeader = (sQMIRawContentHeader*)pOut;
    pHeader->mTypeID = 0x01;
```

```
pHeader->mLength = tlvx01Sz;

ULONG offset = sizeof( sQMIRawContentHeader );

sUIMUnblockPINRequest_SessionInfo * pTLVx01;
pTLVx01 = (sUIMUnblockPINRequest_SessionInfo*)(pOut + offset);
memset( pTLVx01, 0, tlvx01Sz );

// Set the values
if(1)//(IsCdmaTechnology())//TODO:limingjun
{
        pTLVx01->mSessionType = eQMIUIMSessionTypes_Primary1XProvisioning;
}
else
{
        pTLVx01->mSessionType = eQMIUIMSessionTypes_PrimaryGWProvisioning;
}

pTLVx01->mAIDLength = 0;
offset += sizeof( sUIMVerifyPINRequest_SessionInfo );


// Add arguments
std::string oldVal( pPUKValue );
ULONG oldValSz = (ULONG)oldVal.size();
std::string newVal( pNewValue );
ULONG newValSz = (ULONG)newVal.size();

// Check size
WORD tlvx02Sz = sizeof( sUIMUnblockPINRequest_Unblock )
                + (WORD)oldValSz + (WORD)newValSz;
if (*pOutLen < sizeof( sQMIRawContentHeader ) + tlvx02Sz)
{
    return eGOBI_ERR_BUFFER_SZ;
}

pHeader = (sQMIRawContentHeader*)(pOut+offset);
pHeader->mTypeID = 0x02;
pHeader->mLength = tlvx02Sz;
offset += sizeof( sQMIRawContentHeader );

// First part of the TLV
sUIMUnblockPINRequest_Unblock1 * pTLVx02_1;
pTLVx02_1 = (sUIMUnblockPINRequest_Unblock1*)(pOut + offset);
memset( pTLVx02_1, 0, tlvx02Sz );

pTLVx02_1->mID = (eQMIUIMPINIDs)id;
pTLVx02_1->mPUKLength = (UINT8)oldValSz;
offset += sizeof( sUIMUnblockPINRequest_Unblock1 );

// mPUKValue string
memcpy( (pOut + offset), oldVal.c_str(), oldValSz );
offset += oldValSz;

// Second part of the TLV
```

```
    sUIMUnblockPINRequest_Unblock2 * pTLVx02_2;
    pTLVx02_2 = (sUIMUnblockPINRequest_Unblock2*)(pOut + offset);

    pTLVx02_2->mNewPINLength = (UINT8)newValSz;
    offset += sizeof( sUIMUnblockPINRequest_Unblock2 );

    // mNewPINValue string
    memcpy( (pOut + offset), newVal.c_str(), newValSz );
    offset += newValSz;

    *pOutLen = offset;

    return eGOBI_ERR_NONE;
}

ULONG ParseUIMUnblockPIN(
    ULONG                       inLen,
    const BYTE *                pIn,
    ULONG *                     pVerifyRetriesLeft,
    ULONG *                     pUnblockRetriesLeft )
{
    // Validate arguments
    if (pIn == 0 || pVerifyRetriesLeft == 0 || pUnblockRetriesLeft == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Find the TLV
    const sDMSUIMUnblockPINResponse_RetryInfo * pTLVx10;
    ULONG outLenx10;
    ULONG rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
    }

    // Is the TLV large enough?
    if (outLenx10 < sizeof( sDMSUIMUnblockPINResponse_RetryInfo ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    *pVerifyRetriesLeft = pTLVx10->mRemainingVerifyRetries;
    *pUnblockRetriesLeft = pTLVx10->mRemainingUnblockRetries;

    return eGOBI_ERR_NONE;
}


SetUimUnblockPINCB(UimUnblockPINCallback);
    ULONG li = 1024;
    BYTE req[1024] = { 0 };
    ULONG lo = 2048;
    BYTE rsp[2048] = { 0 };
```

```
PackUIMUnblockPIN(&li,req,pINID,pPUK,pNewPIN);
status = mpFnUimUnblockPIN( mhGobi, 2000, li, &req[0],&lo,&rsp[0]);
if (status != 0)
{
        ParseUIMUnblockPIN(lo,&rsp[0],pVerifyRetriesLeft,pUnblockRetriesLeft);
    return status;
}
```

3.1.27. UIMGetCardStatus

Description	Retrieves the current status of the card
Request Structure	

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for UIMGetCardStatus()
	struct sUIMGetCardStatusResponse_Status1
	{
	UINT8 mPrimaryGWSlot;
	UINT8 mPrimaryGWApplication;
	UINT8 mPrimary1XSlot;
	UINT8 mPrimary1XApplication;
	UINT8 mSecondaryGWSlot;
	UINT8 mSecondaryGWApplication;
	UINT8 mSecondary1XSlot;
	UINT8 mSecondary1XApplication;
	UINT8 mSlotsAvailable;
	eQMIUIMCardStates mCardState;
	eQMIUIMPINStates mUPINState;
	UINT8 mRemainingUPINVerifyRetries;
	UINT8 mRemainingUPINUnblockRetries;
	eQMIUIMCardErrorCodes mCardErrorCode;
	UINT8 mApplicationsAvailable;
	eQMIUIMApplicationTypes mApplicationType;
	eQMIUIMApplicationStates mApplicationState;
	eQMIUIMPersonalizationStates mPersonalizationState;
	eQMIUIMPersonalizationFeatures mPersonalizationFeature;
	UINT8 mRemainingPersonalizationVerifyRetries;
	UINT8 mRemainingPersonalizationUnblockRetries;
	UINT8 mAIDLength;

```
    // This array must be the size specified by mAIDLength
    // UINT8 mAID[1];
};

struct sUIMGetCardStatusResponse_Status2
{
    INT8 mUPINReplacesPIN1;
    eQMIUIMPINStates mPIN1State;
    UINT8 mRemainingPIN1VerifyRetries;
    UINT8 mRemainingPIN1UnblockRetries;
    eQMIUIMPINStates mPIN2State;
    UINT8 mRemainingPIN2VerifyRetries;
    UINT8 mRemainingPIN2UnblockRetries;
};

struct sUIMGetCardStatusResponse_Status
{
    sUIMGetCardStatusResponse_Status1 mUIMGetCardStatusResponse_Status1;
    sUIMGetCardStatusResponse_Status2 mUIMGetCardStatusResponse_Status2;
};

// Structure to describe response TLV 0x11 for UIMGetCardStatus()
struct sUIMGetCardStatusResponse_HotSwapStatus
{
    UINT8 mHotSwapLength;

    // This array must be the size specified by mHotSwapLength
    // eQMIUIMHotSwap mHotSwap[1];
};

// Structure to describe response TLV 0x12 for UIMGetCardStatus()
struct sUIMGetCardStatusResponse_ValidCardStatus
{
    UINT8 mCardStatusValidCount;

    // This array must be the size specified by mCardStatusValidCount
```

	// INT8 mCardStatusValid[1]; };
Return Value	QMI_ERR_NONE No error in the request. QMI_ERR_INTERNAL Unexpected error occurred during processing. QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission. QMI_ERR_NO_MEMORY Device could not allocate memory to formulate the response
Constraint	

Example:

```
ULONG ParseUIMGetCardStatus(
    ULONG                           inLen,
    const BYTE *                    pIn,
    eQMIUIMCardStates                       *   cardStates,
    eQMIUIMPINStates                        *   pinStates ,
    UINT8 *pinRetryTimes,
    UINT8 *pinUnblockTimes                      )
{
    // Validate arguments
    if (pIn == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    // Find the TLV
    const sUIMGetCardStatusResponse_Status * pTLVx10;
    ULONG outLenx10;
    ULONG rc = GetTLV( inLen, pIn, 0x10, &outLenx10, (const BYTE **)&pTLVx10 );
    if (rc != eGOBI_ERR_NONE)
    {
        return rc;
    }

    // Is the TLV large enough?
    if (outLenx10 < sizeof( sUIMGetCardStatusResponse_Status ))
    {
        return eGOBI_ERR_MALFORMED_RSP;
    }

    // The TLV only contains the string

    *cardStates = pTLVx10->mUIMGetCardStatusResponse_Status1.mCardState;
    *pinStates = pTLVx10->mUIMGetCardStatusResponse_Status1.mUPINState;
    if(eQMIUIMPINStates_Unknown == *pinStates)
    {
            *pinStates = pTLVx10->mUIMGetCardStatusResponse_Status2.mPIN1State;
            *pinRetryTimes = pTLVx10->mUIMGetCardStatusResponse_Status2.mRemainingPIN1VerifyRet
ries;
            *pinUnblockTimes = pTLVx10->mUIMGetCardStatusResponse_Status2.mRemainingPIN1Unblo
ckRetries;
    }
    else
```

```
    {
            *pinRetryTimes = pTLVx10->mUIMGetCardStatusResponse_Status1.mRemainingUPINVerifyRe
tries;
            *pinUnblockTimes = pTLVx10->mUIMGetCardStatusResponse_Status1.mRemainingUPINUnbl
ockRetries;
    }

    return eGOBI_ERR_NONE;
}

    ULONG lo = 1024;
    BYTE rsp[1024] = { 0 };
    status = UIMGetCardStatus ( mhGobi, 2000, 0, 0, &lo, &rsp[0] );
    if (status != 0)
    {
        return status;
    }

    status = ParseUIMGetCardStatus( lo,
                                    &rsp[0],
                                    cardStates,
                                    pinStates,
                                                    pinRetryTimes,
                                                    pinUnblockTimes);
```

3.1.28. DMSUIMGetICCID

Description	Queries the Integrated Circuit Card ID (ICCID) of the UIM for the device
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for DMSUIMGetICCID()
	struct sDMSUIMGetICCIDResponse_ICCID
	{
	// String is variable length, but must be size of the container
	// char mICCID[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

3.1.29. DMSGetIMSI

Description	Requests the device IMSI

Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for DMSGetIMSI()
	struct sDMSGetIMSIResponse_IMSI
	{
	// String is variable length, but must be size of the container
	// char mIMSI[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

3.1.30. DMSGetDeviceRevision

Description	Requests the device firmware revision identification
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for DMSGetDeviceRevision()
	struct sDMSGetDeviceRevisionResponse_Revision
	{
	// String is variable length, but must be size of the container
	// char mDeviceRevisionID[1];
	};
	Optional TLVs
	// Structure to describe response TLV 0x10 for DMSGetDeviceRevision()
	struct sDMSGetDeviceRevisionResponse_BootCodeRevision
	{
	// String is variable length, but must be size of the container
	// char mBootCodeRevisionID[1];
	};
	
	// Structure to describe response TLV 0x11 for DMSGetDeviceRevision()
	struct sDMSGetDeviceRevisionResponse_UQCNRevision

	{
	// String is variable length, but must be size of the container
	// char mBootCodeRevisionID[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

3.1.31. DMSGetDeviceCapabilities

Description	Requests the device and gets device capabilities
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for DMSGetDeviceCapabilities()
	struct sDMSGetDeviceCapabilitiesResponse_Capabilities
	{
	UINT32 mMaxTXRatebps;
	UINT32 mMaxRXRatebps;
	eQMIDMSDataServiceCapabilities1 mDataServiceCapability;
	INT8 mSIMSupported;
	UINT8 mRadioInterfaceCount;
	
	// This array must be the size specified by mRadioInterfaceCount
	// eQMIDMSRadioInterfaces mRadioInterface[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate the response
Constraint	

Example:

3.1.32. DMSGetDeviceManfacturer

Description	Requests the device the manufacturer information
Request	

Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for DMSGetDeviceManfacturer()
	struct sDMSGetDeviceManfacturerResponse_Manfacturer
	{
	// String is variable length, but must be size of the container
	// char mDeviceManfacturer[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate the response
Constraint	

Example:

3.1.33. DMSGetDeviceModel

Description	Requests the device model identification
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for DMSGetDeviceModel()
	struct sDMSGetDeviceModelResponse_Model
	{
	// String is variable length, but must be size of the container
	// char mDeviceModelID[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

3.1.34. DMSGetHardwareRevision

Description	Queries the hardware revision of the device
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for DMSGetHardwareRevision()
	struct sDMSGetHardwareRevisionResponse_HardwareRevision
	{
	// String is variable length, but must be size of the container
	// char mDeviceHardwareRevision[1];
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

3.1.35. DMSGetDeviceSerialNumbers

Description	Requests the serial numbers of the device
Request Structure	

Response Structure	**Optional TLVs**
	// Structure to describe response TLV 0x10 for DMSGetDeviceSerialNumbers()
	struct sDMSGetDeviceSerialNumbersResponse_ESN
	{
	// String is variable length, but must be size of the container
	// char mESN[1];
	};
	
	// Structure to describe response TLV 0x11 for DMSGetDeviceSerialNumbers()
	struct sDMSGetDeviceSerialNumbersResponse_IMEI
	{
	// String is variable length, but must be size of the container
	// char mIMEI[1];
	};
	
	// Structure to describe response TLV 0x12 for DMSGetDeviceSerialNumbers()
	struct sDMSGetDeviceSerialNumbersResponse_MEID
	{
	// String is variable length, but must be size of the container

	// char mMEID[1]; };
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_NOT_PROVISIONED Device does not support voice service or the value is not provisioned in the device
Constraint	

Example:

3.1.36. DMSUIMChangePIN

Description	Changes the PIN value.

Request Structure	// Structure to describe request TLV 0x01 for DMSUIMChangePIN()
	struct sDMSUIMChangePINRequest_Info1
	{
	UINT8 mPINID;
	UINT8 mOldPINLength;
	
	// This array must be the size specified by mOldPINLength
	// char mOldPINValue[1];
	};
	
	struct sDMSUIMChangePINRequest_Info2
	{
	UINT8 mNewPINLength;
	
	// This array must be the size specified by mNewPINLength
	// char mNewPINValue[1];
	};
	
	struct sDMSUIMChangePINRequest_Info
	{
	sDMSUIMChangePINRequest_Info1 mDMSUIMChangePINRequest_Info1;
	sDMSUIMChangePINRequest_Info2 mDMSUIMChangePINRequest_Info2;
	};

Response Structure	**Mandatory TLVs** // Structure to describe request TLV 0x01 for DMSUIMChangePIN()

	struct sDMSUIMChangePINRequest_Info1
	{
	UINT8 mPINID;
	UINT8 mOldPINLength;
	
	// This array must be the size specified by mOldPINLength
	// char mOldPINValue[1];
	};
	
	struct sDMSUIMChangePINRequest_Info2
	{
	UINT8 mNewPINLength;
	
	// This array must be the size specified by mNewPINLength
	// char mNewPINValue[1];
	};
	
	struct sDMSUIMChangePINRequest_Info
	{
	sDMSUIMChangePINRequest_Info1 mDMSUIMChangePINRequest_Info1;
	sDMSUIMChangePINRequest_Info2 mDMSUIMChangePINRequest_Info2;
	};
	
	// Structure to describe response TLV 0x10 for DMSUIMChangePIN()
	struct sDMSUIMChangePINResponse_RetryInfo
	{
	UINT8 mRemainingVerifyRetries;
	UINT8 mRemainingUnblockRetries;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or message was corrupted during transmission QMI_ERR_NO_EFFECT Operation had no effect QMI_ERR_ARG_TOO_LONG Device cannot handle the length of the PIN specified in the Request

	QMI_ERR_INCORRECT_PIN PIN specified in the request is incorrect QMI_ERR_PIN_BLOCKED PIN is blocked. An unblock operation needs to be issued QMI_ERR_PIN_PERM_BLOCKED PIN is permanently blocked; the SIM is unusable QMI_ERR_UIM_NOT_INITIALIZED PIN is not yet initialized because the SIM initialization has not finished; try the PIN operation later QMI_ERR_OP_DEVICE_UNSUPPORTED Operation is not supported by the device QMI_ERR_MISSING_ARG Some TLV was missing in the request QMI_ERR_INVALID_PINID PIN specified in the request is invalid QMI_ERR_ACCESS_DENIED Operation cannot be performed because the UIM cannot be accessed
Constraint	

Example:

3.1.37. DMSUIMSetPINProtection

Description	Enables or disables protection of UIM contents by a specified PIN.

Request Structure	struct sDMSUIMSetPINProtectionRequest_Info
	{
	UINT8 mPINID;
	UINT8 mPINEnabled;
	UINT8 mPINLength;
	
	// This array must be the size specified by mPINLength
	// char mPINValue[1];
	};

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x10 for DMSUIMSetPINProtection()
	struct sDMSUIMSetPINProtectionResponse_RetryInfo
	{
	UINT8 mRemainingVerifyRetries;
	UINT8 mRemainingUnblockRetries;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control

	point or message was corrupted during transmission QMI_ERR_NO_EFFECT Operation had no effect QMI_ERR_ARG_TOO_LONG Device cannot handle the length of the PIN specified in the Request QMI_ERR_INCORRECT_PIN PIN specified in the request is incorrect QMI_ERR_PIN_BLOCKED PIN is blocked. An unblock operation needs to be issued QMI_ERR_PIN_PERM_BLOCKED PIN is permanently blocked; the SIM is unusable QMI_ERR_UIM_NOT_INITIALIZED PIN is not yet initialized because the SIM initialization has not finished; try the PIN operation later QMI_ERR_OP_DEVICE_UNSUPPORTED Operation is not supported by the device QMI_ERR_MISSING_ARG Some TLV was missing in the request QMI_ERR_INVALID_PINID PIN specified in the request is invalid QMI_ERR_ACCESS_DENIED Operation cannot be performed because the UIM cannot be accessed
Constraint	

Example:

3.1.38. WDSSetEventReport

Description	Sets the wireless data connection state reporting conditions for the requesting control point

Request Structure	**Optional TLVs**
	// Structure to describe request TLV 0x10 for WDSSetEventReport()
	struct sWDSSetEventReportRequest_ChannelRateIndicator
	{
	INT8 mReportChannelRate;
	};
	
	// Structure to describe request TLV 0x11 for WDSSetEventReport()
	struct sWDSSetEventReportRequest_TransferStatisticsIndicator
	{
	UINT8 mTransferStatisticsIntervalSeconds;
	bool mReportTXPacketSuccesses:1;
	bool mReportRXPacketSuccesses:1;
	bool mReportTXPacketErrors:1;
	bool mReportRXPacketErrors:1;

```
    bool mReportTXOverflows:1;
    bool mReportRXOverflows:1;
    bool mTXByteTotal:1;
    bool mRXByteTotal:1;
    bool mTXPacketsDropped:1;
    bool mRXPacketsDropped:1;

    // Padding out 22 bits
    UINT8 mReserved1:6;
    UINT8 mReserved2[2];
};

// Structure to describe request TLV 0x12 for WDSSetEventReport()
struct sWDSSetEventReportRequest_DataBearerTechnologyIndicator
{
    INT8 mReportDataBearerTechnology;
};

// Structure to describe request TLV 0x13 for WDSSetEventReport()
struct sWDSSetEventReportRequest_DormancyStatusIndicator
{
    INT8 mReportDormancyStatus;
};

// Structure to describe request TLV 0x14 for WDSSetEventReport()
struct sWDSSetEventReportRequest_MIPStatusIndicator
{
    INT8 mReportMIPStatus;
};

// Structure to describe request TLV 0x15 for WDSSetEventReport()
struct sWDSSetEventReportRequest_CurrentDataBearerTechnologyIndicator
{
    INT8 mReportDataBearerTechnology;
};

// Structure to describe request TLV 0x17 for WDSSetEventReport()
```

	struct sWDSSetEventReportRequest_DataCallStatusIndicator
	{
	INT8 mReportDataCallStatus;
	};
	
	// Structure to describe request TLV 0x18 for WDSSetEventReport()
	struct sWDSSetEventReportRequest_PreferredDataSystemIndicator
	{
	INT8 mReportPreferredDataSystem;
	};
	
	// Structure to describe request TLV 0x19 for WDSSetEventReport()
	struct sWDSSetEventReportRequest_EVDOPMChangeIndicator
	{
	INT8 mReportEVDOPageMonitorPeriodChange;
	};
	
	// Structure to describe request TLV 0x1A for WDSSetEventReport()
	struct sWDSSetEventReportRequest_DataSystemsIndicator
	{
	INT8 mReportDataSystems;
	};
	
	// Structure to describe request TLV 0x1B for WDSSetEventReport()
	struct sWDSSetEventReportRequest_UplinkFlowControlIndicator
	{
	INT8 mReportUplinkFlowControl;
	};
	
	// Structure to describe request TLV 0x1C for WDSSetEventReport()
	struct sWDSSetEventReportRequest_LimitedDataSystemStatusIndicator
	{
	INT8 mReportLimitedDataSystemStatus;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request

	QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_MISSING_ARG Some TLV was missing in the request
Constraint	
MSG ID	QMI_WDS_EVENT_REPORT_IND 0x0001

IND Structure	// Structure to describe indication TLV 0x10 for WDS EventReport
	struct sWDSEventReportIndication_TXPacketSuccesses
	{
	UINT32 mTXPacketSuccesses;
	};
	
	// Structure to describe indication TLV 0x11 for WDS EventReport
	struct sWDSEventReportIndication_RXPacketSuccesses
	{
	UINT32 mRXPacketSuccesses;
	};
	
	// Structure to describe indication TLV 0x12 for WDS EventReport
	struct sWDSEventReportIndication_TXPacketErrors
	{
	UINT32 mTXPacketErrors;
	};
	
	// Structure to describe indication TLV 0x13 for WDS EventReport
	struct sWDSEventReportIndication_RXPacketErrors
	{
	UINT32 mRXPacketErrors;
	};
	
	// Structure to describe indication TLV 0x14 for WDS EventReport
	struct sWDSEventReportIndication_TXOverflows
	{
	UINT32 mTXOverflows;
	};
	
	// Structure to describe indication TLV 0x15 for WDS EventReport

```
struct sWDSEventReportIndication_RXOverflows
{
    UINT32 mRXOverflows;
};

// Structure to describe indication TLV 0x16 for WDS EventReport
struct sWDSEventReportIndication_ChannelRates
{
    UINT32 mChannelTXRatebps;
    UINT32 mChannelRXRatebps;
};

// Structure to describe indication TLV 0x17 for WDS EventReport
struct sWDSEventReportIndication_DataBearerTechnology
{
    eQMIDataBearerTechnologies mDataBearerTechnology;
};

// Structure to describe indication TLV 0x18 for WDS EventReport
struct sWDSEventReportIndication_DormancyStatus
{
    eQMIDormancyStatus mDormancyStatus;
};

// Structure to describe indication TLV 0x19 for WDS EventReport
struct sWDSEventReportIndication_TXBytes
{
    UINT64 mTXByteTotal;
};

// Structure to describe indication TLV 0x1A for WDS EventReport
struct sWDSEventReportIndication_RXBytes
{
    UINT64 mRXByteTotal;
};

// Structure to describe indication TLV 0x1B for WDS EventReport
```

```
struct sWDSEventReportIndication_MIPStatus
{
    UINT8 mMIPStatus;
};

// Structure to describe indication TLV 0x1D for WDS EventReport
struct sWDSEventReportIndication_CurrentDataBearerTechnology
{
    eQMIWDSNetworkTypes mNetworkType;

    // The following union is based on the value of mNetworkType
    union uValOfNetworkType
    {
        // If the value of mNetworkType == 1
        struct sNetworkTypeIs1
        {
            bool mCDMA1x:1;
            bool mCDMA1xEvDORev0:1;
            bool mCDMA1xEvDORevA:1;
            bool mCDMA1xEvDORevB:1;
            bool mCDMAEHRPD:1;
            bool mCDMAFMC:1;

            // Padding out 25 bits
            UINT8 mReserved1:2;
            UINT8 mReserved2[2];
            UINT8 mReserved3:7;

            bool mNullBearer:1;

            // The following union is for handing both mCDMA1x and
mCDMA1xEvDORev(0, A, B)
            union uValOfCDMA1x_or_CDMA1xEvDORevX
            {
                // If the value of mCDMA1x == 1
                struct sCDMA1xIs1
                {
```

```
        bool mCDMA1xIS95:1;
        bool mCDMA1xIS2000:1;
        bool mCDMA1xIS2000RelA:1;

        // Padding out 29 bits
        UINT8 mReserved4:5;
        UINT8 mReserved5[3];
    };

    sCDMA1xIs1 mCDMA1xIs1;

    // If the value of mCDMA1xEvDORev0 == 1
    struct sCDMA1xEvDORev0Is1
    {
        bool mCDMA1xEvDORev0DPA:1;

        // Padding out 31 bits
        UINT8 mReserved6:7;
        UINT8 mReserved7[3];
    };

    sCDMA1xEvDORev0Is1 mCDMA1xEvDORev0Is1;

    // If the value of mCDMA1xEvDORevA == 1
    struct sCDMA1xEvDORevAIs1
    {
        bool mCDMA1xEvDORevADPA:1;
        bool mCDMA1xEvDORevAMFPA:1;
        bool mCDMA1xEvDORevAEMPA:1;
        bool mCDMA1xEvDORevAEMPAEHRPD:1;

        // Padding out 28 bits
        UINT8 mReserved8:4;
        UINT8 mReserved9[3];
    };

    sCDMA1xEvDORevAIs1 mCDMA1xEvDORevAIs1;
```

```
                // If the value of mCDMA1xEvDORevB == 1
                struct sCDMA1xEvDORevBIs1
                {
                    bool mCDMA1xEvDORevBDPA:1;
                    bool mCDMA1xEvDORevBMFPA:1;
                    bool mCDMA1xEvDORevBEMPA:1;
                    bool mCDMA1xEvDORevBEMPAEHRPD:1;
                    bool mCDMA1xEvDORevBMMPA:1;
                    bool mCDMA1xEvDORevBMMPAEHRPD:1;

                    // Padding out 26 bits
                    UINT8 mReserved10:2;
                    UINT8 mReserved11[3];
                };

                sCDMA1xEvDORevBIs1 mCDMA1xEvDORevBIs1;

                // Padding out 32 bits
                UINT8 mReserved12[4];
            };

            uValOfCDMA1x_or_CDMA1xEvDORevX
mValOfCDMA1x_or_CDMA1xEvDORevX;
        };

        sNetworkTypeIs1 mNetworkTypeIs1;

        // If the value of mNetworkType == 2
        struct sNetworkTypeIs2
        {
            bool mWCDMA:1;
            bool mGPRS:1;
            bool mHSDPA:1;
            bool mHSUPA:1;
            bool mEDGE:1;
            bool mLTE:1;
```

```
            bool mHSDPAPlus:1;
            bool mDualCellHSDPAPlus:1;
            bool m64QAM:1;
            bool mTDSCDMA:1;

            // Padding out 21 bits
            UINT8 mReserved13:6;
            UINT8 mReserved14;
            UINT8 mReserved15:7;

            bool mNullBearer:1;
        };

        sNetworkTypeIs2 mNetworkTypeIs2;

        // Padding out 64 bits
        UINT8 mReserved16[8];
    };

    uValOfNetworkType mValOfNetworkType;
};

// Structure to describe indication TLV 0x1F for WDS EventReport
struct sWDSEventReportIndication_DataCallStatus
{
    eQMIWDSDataCallStatus mDataCallStatus;
};

// Structure to describe indication TLV 0x20 for WDS EventReport
struct sWDSEventReportIndication_PreferredDataSystem
{
    eQMIWDSDataSystems mPreferredDataSystem;
};

// Structure to describe indication TLV 0x22 for WDS EventReport
struct sWDSEventReportIndication_DataCallType
{
```

```
    eQMIWDSDataCallTypes mDataCallType;
    eQMIWDSTetheredCallTypes mTetheredCallType;
};

// Structure to describe indication TLV 0x23 for WDS EventReport
struct sWDSEventReportIndication_EVDOPageMonitorPeriodChange
{
    UINT8 mEVDOPageMonitorPeriodChange;
    INT8 mEVDOForceLongSleep;
};

// Structure to describe indication TLV 0x24 for WDS EventReport
struct sWDSEventReportIndication_DataSystems
{
    eQMIWDSDataSystemNetworkTypes mPreferredNetworkType;
    UINT8 mNetworkCount;

    struct sNetwork
    {
        eQMIWDSDataSystemNetworkTypes mNetworkType;

        // The following union is based on the value of mNetworkType
        union uValOfNetworkType
        {
            // If the value of mNetworkType == 0
            struct sNetworkTypeIs0
            {
                bool mWCDMA:1;
                bool mGPRS:1;
                bool mHSDPA:1;
                bool mHSUPA:1;
                bool mEDGE:1;
                bool mLTE:1;
                bool mHSDPAPlus:1;
                bool mDualCellHSDPAPlus:1;
                bool m64QAM:1;
                bool mTDSCDMA:1;
```

```
    // Padding out 21 bits
    UINT8 mReserved1:6;
    UINT8 mReserved2;
    UINT8 mReserved3:7;

    bool mNULLBearer:1;
};

sNetworkTypeIs0 mNetworkTypeIs0;

// If the value of mNetworkType == 1
struct sNetworkTypeIs1
{
    bool mCDMA1x:1;
    bool mCDMA1xEvDORev0:1;
    bool mCDMA1xEvDORevA:1;
    bool mCDMA1xEvDORevB:1;
    bool mCDMAEHRPD:1;
    bool mCDMAFMC:1;

    // Padding out 25 bits
    UINT8 mReserved4:2;
    UINT8 mReserved5[2];
    UINT8 mReserved6:7;

    bool mNULLBearer:1;

    // The following union is for handing all mCDMA1x types
    union uValOfCDMA1xTypes
    {
        // If the value of mCDMA1x == 1
        struct sCDMA1xIs1
        {
            bool mCDMA1xIS95:1;
            bool mCDMA1xIS2000:1;
            bool mCDMA1xIS2000RelA:1;
```

```
    // Padding out 29 bits
    UINT8 mReserved7:5;
    UINT8 mReserved8[3];
};

sCDMA1xIs1 mCDMA1xIs1;

// If the value of mCDMA1xEvDORev0 == 1
struct sCDMA1xEvDORev0Is1
{
    bool mCDMA1xEvDORev0DPA:1;

    // Padding out 31 bits
    UINT8 mReserved9:7;
    UINT8 mReserved10[3];
};

sCDMA1xEvDORev0Is1 mCDMA1xEvDORev0Is1;

// If the value of mCDMA1xEvDORevA == 1
struct sCDMA1xEvDORevAIs1
{
    bool mCDMA1xEvDORevADPA:1;
    bool mCDMA1xEvDORevAMFPA:1;
    bool mCDMA1xEvDORevAEMPA:1;
    bool mCDMA1xEvDORevAEMPAEHRPD:1;

    // Padding out 28 bits
    UINT8 mReserved11:4;
    UINT8 mReserved12[3];
};

sCDMA1xEvDORevAIs1 mCDMA1xEvDORevAIs1;

// If the value of mCDMA1xEvDORevB == 1
struct sCDMA1xEvDORevBIs1
```

```
                    {
                        bool mCDMA1xEvDORevBDPA:1;
                        bool mCDMA1xEvDORevBMFPA:1;
                        bool mCDMA1xEvDORevBEMPA:1;
                        bool mCDMA1xEvDORevBEMPAEHRPD:1;
                        bool mCDMA1xEvDORevBMMPA:1;
                        bool mCDMA1xEvDORevBMMPAEHRPD:1;

                        // Padding out 26 bits
                        UINT8 mReserved13:2;
                        UINT8 mReserved14[3];
                    };

                    sCDMA1xEvDORevBIs1 mCDMA1xEvDORevBIs1;

                    // Padding out 32 bits
                    UINT8 mReserved15[4];
                };

                uValOfCDMA1xTypes mValOfCDMA1xTypes;
            };

            sNetworkTypeIs1 mNetworkTypeIs1;

            // Padding out 64 bits
            UINT8 mReserved16[8];
        };

        uValOfNetworkType mValOfNetworkType;
    };

    // This array must be the size specified by mNetworkCount
    // sNetwork mNetworks[1];
};

// Structure to describe indication TLV 0x25 for WDS EventReport
struct sWDSEventReportIndication_TXPacketsDropped
```

```
{
    UINT32 mTXPacketsDropped;
};

// Structure to describe indication TLV 0x26 for WDS EventReport
struct sWDSEventReportIndication_RXPacketsDropped
{
    UINT32 mRXPacketsDropped;
};

// Structure to describe indication TLV 0x27 for WDS EventReport
struct sWDSEventReportIndication_UplinkFlowControl
{
    INT8 mUplinkFlowControlEnabled;
};

// Structure to describe indication TLV 0x28 for WDS EventReport
struct sWDSEventReportIndication_DataCallAddressFamily
{
    eQMIWDSAddressFamilies mAddressFamily;
};
```

Example:

```
void __cdecl WDSEventReportCallback(
    ULONG                           svcID,
    ULONG                           msgID,
    GOBIHANDLE                       /* handle */,
    ULONG                           outLen,
    const BYTE *                    pOut )
{
    //C2Win * pMainWnd = (C2Win *)gApp.GetMainWnd();
    if (svcID != 1 || msgID != 1)
    {
        return;
    }

    std::map <UINT8, const sQMIRawContentHeader *> tlvs = GetTLVs( &pOut[0], outLen );
    std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x17 );
    if (pIter != tlvs.end())
    {
        const sQMIRawContentHeader * pTmp = pIter->second;
        if (pTmp->mLength >= sizeof (sWDSEventReportIndication_DataBearerTechnology))
        {
            pTmp++;
            const sWDSEventReportIndication_DataBearerTechnology * pDBT =
```

```
            (const sWDSEventReportIndication_DataBearerTechnology *)pTmp;
                pTheDialog->PostMessage(WM_TECHNOLOGY_NOTIFY, (WPARAM)pDBT->mDataB
earerTechnology, 0);

        }
    }

    pIter = tlvs.find( 0x16 );
    if (pIter != tlvs.end())
    {
        const sQMIRawContentHeader * pTmp = pIter->second;
        if (pTmp->mLength >= sizeof (sWDSEventReportIndication_ChannelRates))
        {
            pTmp++;
            const sWDSEventReportIndication_ChannelRates * pDBC =
                (const sWDSEventReportIndication_ChannelRates *)pTmp;
                    pTheDialog->PostMessage(WM_RATE_NOTIFY, (WPARAM)pDBC->mChannelTXRateb
ps, pDBC->mChannelRXRatebps);

        }
    }

    ULONGLONG txTotalBytes = ULLONG_MAX;
    ULONGLONG rxTotalBytes = ULLONG_MAX;

    pIter = tlvs.find( 0x19 );
    if (pIter != tlvs.end())
    {
        const sQMIRawContentHeader * pTmp = pIter->second;
        if (pTmp->mLength >= sizeof (sWDSEventReportIndication_TXBytes))
        {
            pTmp++;
            const sWDSEventReportIndication_TXBytes * pTX =
                (const sWDSEventReportIndication_TXBytes *)pTmp;

            txTotalBytes = pTX->mTXByteTotal;
        }
    }

    pIter = tlvs.find( 0x1A );
    if (pIter != tlvs.end())
    {
        const sQMIRawContentHeader * pTmp = pIter->second;
        if (pTmp->mLength >= sizeof (sWDSEventReportIndication_RXBytes))
        {
            pTmp++;
            const sWDSEventReportIndication_RXBytes * pRX =
                (const sWDSEventReportIndication_RXBytes *)pTmp;

            rxTotalBytes = pRX->mRXByteTotal;
        }
    }

    if (txTotalBytes != ULLONG_MAX || rxTotalBytes != ULLONG_MAX)
    {
```

```
        sByteTotals * pByteTotals = new sByteTotals();
        pByteTotals->mTXTotal = txTotalBytes;
        pByteTotals->mRXTotal = rxTotalBytes;
            pTheDialog->PostMessage(WM_USER_BYTE_TOTALS_NOTIFY, (WPARAM)pByteTotals, 0);
    }
}

ULONG cGobiCMDLL::SetWDSEventReportCB(
    tFNGenericCallback          pCallback,
    BYTE                        interval )
{
    // Set WDS event callback?
    ULONG status = 1;
    if ( (mpWDSSetEventReport == 0)
    ||    (mpFnSetGenericCallback == 0)
    ||    (mhGobi == 0) )
    {
        return status;
    }

    // Configure the QMI service
    UINT8 req[1024] = { 0 };
    UINT8 * pData = (UINT8 *)&req[0];

    sQMIRawContentHeader * pTLV = (sQMIRawContentHeader *)pData;
     pTLV->mTypeID = 0x10;
    pTLV->mLength = (UINT16)sizeof( sWDSSetEventReportRequest_ChannelRateIndicator );
    pData += sizeof( sQMIRawContentHeader );
    sWDSSetEventReportRequest_ChannelRateIndicator * pTC =
        (sWDSSetEventReportRequest_ChannelRateIndicator *)pData;
    pTC->mReportChannelRate = 1;
    pData += sizeof( sWDSSetEventReportRequest_ChannelRateIndicator );

     pTLV = (sQMIRawContentHeader *)pData;
    pTLV->mTypeID = 0x11;
    pTLV->mLength = (UINT16)sizeof( sWDSSetEventReportRequest_TransferStatisticsIndicator );
    pData += sizeof( sQMIRawContentHeader );

    sWDSSetEventReportRequest_TransferStatisticsIndicator * pTS =
        (sWDSSetEventReportRequest_TransferStatisticsIndicator *)pData;
    pTS->mTransferStatisticsIntervalSeconds = interval;
    pTS->mReportTXPacketSuccesses = 0;
    pTS->mReportRXPacketSuccesses = 0;
    pTS->mReportTXPacketErrors = 0;
    pTS->mReportRXPacketErrors = 0;
    pTS->mReportTXOverflows = 0;
    pTS->mReportRXOverflows = 0;
    pTS->mTXByteTotal = 1;
    pTS->mRXByteTotal = 1;
    pData += sizeof( sWDSSetEventReportRequest_TransferStatisticsIndicator );

    pTLV = (sQMIRawContentHeader *)pData;
    pTLV->mTypeID = 0x12;
    pTLV->mLength = (UINT16)sizeof( sWDSSetEventReportRequest_DataBearerTechnologyIndicator );
```

```
    pData += sizeof( sQMIRawContentHeader );

    sWDSSetEventReportRequest_DataBearerTechnologyIndicator * pTI =
        (sWDSSetEventReportRequest_DataBearerTechnologyIndicator *)pData;
    pTI->mReportDataBearerTechnology = 1;
    pData += sizeof( sWDSSetEventReportRequest_DataBearerTechnologyIndicator );

    ULONG li = (ULONG)pData - (ULONG)&req[0];
    status = mpWDSSetEventReport( mhGobi, 2000, li, &req[0], 0, 0 );
    if (status != 0)
    {
        return status;
    }

    // Configure the callback with the API
    status = mpFnSetGenericCallback( mhGobi, 1, 1, pCallback );
    return status;
}
```

3.1.39. WDSStartNetworkInterface

Description	Activates a packet data session (if not already started) on behalf of the requesting control point

Request Structure	**Optional TLVs**
	// Structure to describe request TLV 0x10 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceRequest_PrimaryDNS
	{
	UINT8 mIPV4Address[4];
	};
	
	// Structure to describe request TLV 0x11 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceRequest_SecondaryDNS
	{
	UINT8 mIPV4Address[4];
	};
	
	// Structure to describe request TLV 0x12 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceRequest_PrimaryNBNS
	{
	UINT8 mIPV4Address[4];
	};
	
	// Structure to describe request TLV 0x13 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceRequest_SecondaryNBNS

```
{
    UINT8 mIPV4Address[4];
};

// Structure to describe request TLV 0x14 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_ContextAPNName
{
    // String is variable length, but must be size of the container
    // char mAPNName[1];
};

// Structure to describe request TLV 0x15 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_IPAddress
{
    UINT8 mIPV4Address[4];
};

// Structure to describe request TLV 0x16 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_Authentication
{
    bool mEnablePAP:1;
    bool mEnableCHAP:1;

    // Padding out 6 bits
    UINT8 mReserved1:6;
};

// Structure to describe request TLV 0x17 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_Username
{
    // String is variable length, but must be size of the container
    // char mUsername[1];
};

// Structure to describe request TLV 0x18 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_Password
{
```

```
    // String is variable length, but must be size of the container
    // char mPassword[1];
};

// Structure to describe request TLV 0x19 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_IPFamily
{
    eQMIWDSIPFamilies mIPFamily;
};

// Structure to describe request TLV 0x30 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_TechnologyPreference
{
    bool mEnable3GPP:1;
    bool mEnable3GPP2:1;

    // Padding out 6 bits
    UINT8 mReserved1:6;
};

// Structure to describe request TLV 0x31 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_3GPPProfileIdentifier
{
    UINT8 mProfileIndex;
};

// Structure to describe request TLV 0x32 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_3GPP2ProfileIdentifier
{
    UINT8 mProfileIndex;
};

// Structure to describe request TLV 0x33 for WDSStartNetworkInterface()
struct sWDSStartNetworkInterfaceRequest_Autoconnect
{
    eQMIWDSAutoconnectSettings mAutoconnectSetting;
};
```

	// Structure to describe request TLV 0x34 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceRequest_ExtendedTechnologyPreference
	{
	eQMIWDSExtendedTechPrefs mExtendedTechnologyPreference;
	};
	
	// Structure to describe request TLV 0x35 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceRequest_CallType
	{
	eQMIWDSCallTypes mCallType;
	};

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceResponse_PacketDataHandle
	{
	UINT32 mPacketDataHandle;
	};
	Optional TLVs
	// Structure to describe response TLV 0x10 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceResponse_CallEndReason
	{
	eQMICallEndReasons mCallEnd;
	};
	
	// Structure to describe response TLV 0x11 for WDSStartNetworkInterface()
	struct sWDSStartNetworkInterfaceResponse_VerboseCallEndReason
	{
	eQMIWDSCallEndReasonTypes mCallEndReasonType;
	
	// The following union is based on the value of mCallEndReasonType
	union uValOfCallEndReasonType
	{
	// Always present
	UINT16 mCallEndReasonValue;

```
// If the value of mCallEndReasonType == 1
struct sCallEndReasonTypeIs1
{
    eQMIWDSMobileIPCallEndReasons mMobileIPCallEndReason;
};

sCallEndReasonTypeIs1 mCallEndReasonTypeIs1;

// If the value of mCallEndReasonType == 2
struct sCallEndReasonTypeIs2
{
    eQMIWDSInternalCallEndReasons mInternalCallEndReason;
};

sCallEndReasonTypeIs2 mCallEndReasonTypeIs2;

// If the value of mCallEndReasonType == 3
struct sCallEndReasonTypeIs3
{
    eQMIWDSCallManagerCallEndReasons mCallManagerCallEndReason;
};

sCallEndReasonTypeIs3 mCallEndReasonTypeIs3;

// If the value of mCallEndReasonType == 6
struct sCallEndReasonTypeIs6
{
    eQMIWDS3GPPCallEndReasons m3GPPCallEndReason;
};

sCallEndReasonTypeIs6 mCallEndReasonTypeIs6;

// If the value of mCallEndReasonType == 7
struct sCallEndReasonTypeIs7
{
    eQMIWDSPPPCallEndReason mPPPCallEndReason;
};
```

```
        sCallEndReasonTypeIs7 mCallEndReasonTypeIs7;

        // If the value of mCallEndReasonType == 8
        struct sCallEndReasonTypeIs8
        {
            eQMIWDSEHRPDCallEndReason mEHRPDCallEndReason;
        };

        sCallEndReasonTypeIs8 mCallEndReasonTypeIs8;

        // If the value of mCallEndReasonType == 9
        struct sCallEndReasonTypeIs9
        {
            eQMIWDSIPv6CallEndReason mIPv6CallEndReason;
        };

        sCallEndReasonTypeIs9 mCallEndReasonTypeIs9;

        // Padding out 16 bits
        UINT8 mReserved1[2];
    };

    uValOfCallEndReasonType mValOfCallEndReasonType;
};
```

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_ARG_TOO_LONG Argument passed in a TLV is larger than the available storage in the device QMI_ERR_INVALID_PROFILE Specified configured profile index does not exist QMI_ERR_NO_EFFECT Control point has already started the network interface QMI_ERR_CALL_FAILED Data call failed QMI_ERR_INVALID_TECH_PREF Invalid technology preference QMI_ERR_INVALID_PDP_TYPE Invalid PDP type

	QMI_ERR_ACCESS_DENIED Autoconnect feature is unavailable at this time QMI_ERR_INVALID_IP_FAMILY_ PREF Invalid IP family preference
Constraint	

Example:

```
ULONG PackStartDataSession(
    ULONG *                         pOutLen,
    BYTE *                          pOut,
    ULONG *                         pTechnology,
    ULONG *                         pPrimaryDNS,
    ULONG *                         pSecondaryDNS,
    ULONG *                         pPrimaryNBNS,
    ULONG *                         pSecondaryNBNS,
    CHAR *                          pAPNName,
    ULONG *                         pIPAddress,
    ULONG *                         pAuthentication,
    CHAR *                          pUsername,
    CHAR *                          pPassword )
{
    // Validate arguments
    if (pOut == 0)
    {
        return eGOBI_ERR_INVALID_ARG;
    }

    sQMIRawContentHeader * pHeader;
    ULONG offset = 0;

    // Add technology, if specified
    if (pTechnology != 0)
    {
        // Check size
        WORD tlvx30Sz = sizeof( sWDSStartNetworkInterfaceRequest_TechnologyPreference );
        if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx30Sz)
        {
            return eGOBI_ERR_BUFFER_SZ;
        }

        pHeader = (sQMIRawContentHeader*)(pOut + offset);
        pHeader->mTypeID = 0x30;
        pHeader->mLength = tlvx30Sz;

        offset += sizeof( sQMIRawContentHeader );

        sWDSStartNetworkInterfaceRequest_TechnologyPreference * pTLVx30;
        pTLVx30 = (sWDSStartNetworkInterfaceRequest_TechnologyPreference*)(pOut + offset);
        memset( pTLVx30, 0, tlvx30Sz );

        // Set the value
        pTLVx30->mEnable3GPP = ((*pTechnology & 0x00000001) != 0);
        pTLVx30->mEnable3GPP2 = ((*pTechnology & 0x00000002) != 0);
```

```
    offset += tlvx30Sz;
}

// Add Primary DNS, if specified
if ((pPrimaryDNS != 0)&&(*pPrimaryDNS != 0))
{
    // Check size
    WORD tlvx10Sz = sizeof( sWDSStartNetworkInterfaceRequest_PrimaryDNS );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx10Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x10;
    pHeader->mLength = tlvx10Sz;

    offset += sizeof( sQMIRawContentHeader );

    sWDSStartNetworkInterfaceRequest_PrimaryDNS * pTLVx10;
    pTLVx10 = (sWDSStartNetworkInterfaceRequest_PrimaryDNS*)(pOut + offset);
    memset( pTLVx10, 0, tlvx10Sz );

    ULONG ip0 = (*pPrimaryDNS & 0x000000FF);
    ULONG ip1 = (*pPrimaryDNS & 0x0000FF00) >> 8;
    ULONG ip2 = (*pPrimaryDNS & 0x00FF0000) >> 16;
    ULONG ip3 = (*pPrimaryDNS & 0xFF000000) >> 24;

    // Set the value
    pTLVx10->mIPV4Address[0] = (INT8)ip0;
    pTLVx10->mIPV4Address[1] = (INT8)ip1;
    pTLVx10->mIPV4Address[2] = (INT8)ip2;
    pTLVx10->mIPV4Address[3] = (INT8)ip3;

    offset += tlvx10Sz;
}

// Add Secondary DNS, if specified
if ((pSecondaryDNS != 0)&&(*pSecondaryDNS != 0))
{
    // Check size
    WORD tlvx11Sz = sizeof( sWDSStartNetworkInterfaceRequest_SecondaryDNS );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx11Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x11;
    pHeader->mLength = tlvx11Sz;

    offset += sizeof( sQMIRawContentHeader );

    sWDSStartNetworkInterfaceRequest_SecondaryDNS * pTLVx11;
```

```
    pTLVx11 = (sWDSStartNetworkInterfaceRequest_SecondaryDNS*)(pOut + offset);
    memset( pTLVx11, 0, tlvx11Sz );

    ULONG ip0 = (*pSecondaryDNS & 0x000000FF);
    ULONG ip1 = (*pSecondaryDNS & 0x0000FF00) >> 8;
    ULONG ip2 = (*pSecondaryDNS & 0x00FF0000) >> 16;
    ULONG ip3 = (*pSecondaryDNS & 0xFF000000) >> 24;

    // Set the value
    pTLVx11->mIPV4Address[0] = (INT8)ip0;
    pTLVx11->mIPV4Address[1] = (INT8)ip1;
    pTLVx11->mIPV4Address[2] = (INT8)ip2;
    pTLVx11->mIPV4Address[3] = (INT8)ip3;

    offset += tlvx11Sz;
}

// Add Primary NBNS, if specified
if (pPrimaryNBNS != 0)
{
    // Check size
    WORD tlvx12Sz = sizeof( sWDSStartNetworkInterfaceRequest_PrimaryNBNS );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx12Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x12;
    pHeader->mLength = tlvx12Sz;

    offset += sizeof( sQMIRawContentHeader );

    sWDSStartNetworkInterfaceRequest_PrimaryNBNS * pTLVx12;
    pTLVx12 = (sWDSStartNetworkInterfaceRequest_PrimaryNBNS*)(pOut + offset);
    memset( pTLVx12, 0, tlvx12Sz );

    ULONG ip0 = (*pPrimaryNBNS & 0x000000FF);
    ULONG ip1 = (*pPrimaryNBNS & 0x0000FF00) >> 8;
    ULONG ip2 = (*pPrimaryNBNS & 0x00FF0000) >> 16;
    ULONG ip3 = (*pPrimaryNBNS & 0xFF000000) >> 24;

    // Set the value
    pTLVx12->mIPV4Address[0] = (INT8)ip0;
    pTLVx12->mIPV4Address[1] = (INT8)ip1;
    pTLVx12->mIPV4Address[2] = (INT8)ip2;
    pTLVx12->mIPV4Address[3] = (INT8)ip3;

    offset += tlvx12Sz;
}

// Add Secondary NBNS, if specified
if (pSecondaryNBNS != 0)
{
    // Check size
```

```
    WORD tlvx13Sz = sizeof( sWDSStartNetworkInterfaceRequest_SecondaryNBNS );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx13Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x13;
    pHeader->mLength = tlvx13Sz;

    offset += sizeof( sQMIRawContentHeader );

    sWDSStartNetworkInterfaceRequest_SecondaryNBNS * pTLVx13;
    pTLVx13 = (sWDSStartNetworkInterfaceRequest_SecondaryNBNS*)(pOut + offset);
    memset( pTLVx13, 0, tlvx13Sz );

    ULONG ip0 = (*pSecondaryNBNS & 0x000000FF);
    ULONG ip1 = (*pSecondaryNBNS & 0x0000FF00) >> 8;
    ULONG ip2 = (*pSecondaryNBNS & 0x00FF0000) >> 16;
    ULONG ip3 = (*pSecondaryNBNS & 0xFF000000) >> 24;

    // Set the value
    pTLVx13->mIPV4Address[0] = (INT8)ip0;
    pTLVx13->mIPV4Address[1] = (INT8)ip1;
    pTLVx13->mIPV4Address[2] = (INT8)ip2;
    pTLVx13->mIPV4Address[3] = (INT8)ip3;

    offset += tlvx13Sz;
}

// Add APN Name, if specified
if ((pAPNName != 0)&&(pAPNName[0] != '\0'))
{
    std::string apnName( pAPNName );

    // Check size
    WORD tlvx14Sz = (WORD)apnName.size();
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx14Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x14;
    pHeader->mLength = tlvx14Sz;

    offset += sizeof( sQMIRawContentHeader );

    // Set the value
    memcpy( (pOut + offset), apnName.c_str(), apnName.size() );

    offset += tlvx14Sz;
}

// Add IP Address, if specified
```

```
if ((pIPAddress != 0)&&(*pIPAddress != 0))
{
    // Check size
    WORD tlvx15Sz = sizeof( sWDSStartNetworkInterfaceRequest_IPAddress );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx15Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x15;
    pHeader->mLength = tlvx15Sz;

    offset += sizeof( sQMIRawContentHeader );

    sWDSStartNetworkInterfaceRequest_IPAddress * pTLVx15;
    pTLVx15 = (sWDSStartNetworkInterfaceRequest_IPAddress*)(pOut + offset);
    memset( pTLVx15, 0, tlvx15Sz );

    ULONG ip0 = (*pIPAddress & 0x000000FF);
    ULONG ip1 = (*pIPAddress & 0x0000FF00) >> 8;
    ULONG ip2 = (*pIPAddress & 0x00FF0000) >> 16;
    ULONG ip3 = (*pIPAddress & 0xFF000000) >> 24;

    // Set the value
    pTLVx15->mIPV4Address[0] = (INT8)ip0;
    pTLVx15->mIPV4Address[1] = (INT8)ip1;
    pTLVx15->mIPV4Address[2] = (INT8)ip2;
    pTLVx15->mIPV4Address[3] = (INT8)ip3;

    offset += tlvx15Sz;
}

// Add Authentication, if specified
if (pAuthentication != 0)
{
    // Check size
    WORD tlvx16Sz = sizeof( sWDSStartNetworkInterfaceRequest_Authentication );
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx16Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x16;
    pHeader->mLength = tlvx16Sz;

    offset += sizeof( sQMIRawContentHeader );

    sWDSStartNetworkInterfaceRequest_Authentication * pTLVx16;
    pTLVx16 = (sWDSStartNetworkInterfaceRequest_Authentication*)(pOut + offset);
    memset( pTLVx16, 0, tlvx16Sz );

    // Set the value
    pTLVx16->mEnablePAP = ((*pAuthentication & 0x00000001) != 0);
```

```
    pTLVx16->mEnableCHAP = ((*pAuthentication & 0x00000002) != 0);

    offset += tlvx16Sz;
}

// Add Username, if specified
if ((pUsername != 0)&&(pUsername[0] != '\0'))
{
    std::string username( pUsername );

    // Check size
    WORD tlvx17Sz = (WORD)username.size();
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx17Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x17;
    pHeader->mLength = tlvx17Sz;

    offset += sizeof( sQMIRawContentHeader );

    // Set the value
    memcpy( (pOut + offset), username.c_str(), username.size() );

    offset += tlvx17Sz;
}

// Add Password, if specified
if ((pPassword != 0)&&(pPassword[0] != '\0'))
{
    std::string password( pPassword );

    // Check size
    WORD tlvx18Sz = (WORD)password.size();
    if (*pOutLen < offset + sizeof( sQMIRawContentHeader ) + tlvx18Sz)
    {
        return eGOBI_ERR_BUFFER_SZ;
    }

    pHeader = (sQMIRawContentHeader*)(pOut + offset);
    pHeader->mTypeID = 0x18;
    pHeader->mLength = tlvx18Sz;

    offset += sizeof( sQMIRawContentHeader );

    // Set the value
    memcpy( (pOut + offset), password.c_str(), password.size() );

    offset += tlvx18Sz;
}

*pOutLen = offset;
```

```
    return eGOBI_ERR_NONE;
}




    ULONG li = 1024;
    ULONG uTechnology = 0;
    ULONG uPrimaryDNS;
    ULONG uSecondDNS;
    ULONG uIPAddress;

    uTechnology |= 0x00000001;

    uPrimaryDNS = g_CurApplyProfConfig.adv.PrimaryDNS.a & 0x000000FF;
    uPrimaryDNS |= (g_CurApplyProfConfig.adv.PrimaryDNS.b<<8) & 0x0000FF00;
    uPrimaryDNS |= (g_CurApplyProfConfig.adv.PrimaryDNS.c<<16) & 0x00FF0000;
    uPrimaryDNS |= (g_CurApplyProfConfig.adv.PrimaryDNS.d<<24) & 0xFF000000;

    uSecondDNS = g_CurApplyProfConfig.adv.SecondaryDNS.a & 0x000000FF;
    uSecondDNS |= (g_CurApplyProfConfig.adv.SecondaryDNS.b<<8) & 0x0000FF00;
    uSecondDNS |= (g_CurApplyProfConfig.adv.SecondaryDNS.c<<16) & 0x00FF0000;
    uSecondDNS |= (g_CurApplyProfConfig.adv.SecondaryDNS.d<<24) & 0xFF000000;

    uIPAddress = g_CurApplyProfConfig.adv.StaticIP.a & 0x000000FF;
    uIPAddress |= (g_CurApplyProfConfig.adv.StaticIP.b<<8) & 0x0000FF00;
    uIPAddress |= (g_CurApplyProfConfig.adv.StaticIP.c<<16) & 0x00FF0000;
    uIPAddress |= (g_CurApplyProfConfig.adv.StaticIP.d<<24) & 0xFF000000;

    PackStartDataSession(&li,req,&uTechnology,&uPrimaryDNS,&uSecondDNS,NULL,NULL,pAPN,&uIPAd
dress,
            (ULONG *)&g_CurApplyProfConfig.adv.nAuthType,g_CurApplyProfConfig.gen.pusername,g_C
urApplyProfConfig.gen.ppassword);
    ULONG lo = 1024;
    BYTE rsp[1024] = { 0 };
    status = WDSStartNetworkInterface ( mhGobi, 300000, li, &req[0], &lo, &rsp[0] );
    if (status != 0)
    {
        return status;
    }

    std::map <UINT8, const sQMIRawContentHeader *> tlvs = GetTLVs( &rsp[0], lo );
    std::map <UINT8, const sQMIRawContentHeader *>::const_iterator pIter = tlvs.find( 0x01 );
    if (pIter == tlvs.end())
    {
        status = 1;
        return status;
    }

    const sQMIRawContentHeader * pTmp = pIter->second;
    if (pTmp->mLength < sizeof (sWDSStartNetworkInterfaceResponse_PacketDataHandle))
    {
        status = 1;
        return status;
```

```
}

pTmp++;
const sWDSStartNetworkInterfaceResponse_PacketDataHandle * pPDH =
    (const sWDSStartNetworkInterfaceResponse_PacketDataHandle *)pTmp;

if (pSessionID != 0)
{
    *pSessionID = pPDH->mPacketDataHandle;
}
```

3.1.40. WDSStopNetworkInterface

Description	Deactivates a packet data session (unless in use by other control points) on behalf of the requesting control point.

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for WDSStopNetworkInterface()
	struct sWDSStopNetworkInterfaceRequest_PacketDataHandle
	{
	UINT32 mPacketDataHandle;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for WDSStopNetworkInterface()
	struct sWDSStopNetworkInterfaceRequest_Autoconnect
	{
	INT8 mAutoconnectOff;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_MISSING_ARG Some TLV is missing QMI_ERR_INVALID_HANDLE Packet_data_handle provided in the request is not valid, i.e., it is not assigned to the control point
Constraint	

Example:

3.1.41. WDSGetPacketServiceStatus

Description	Queries the current packet data connection status
Request Structure	

Response Structure	// Structure to describe response TLV 0x01 for WDSGetPacketServiceStatus()
	struct sWDSGetPacketServiceStatusResponse_Status
	{
	eQMIConnectionStatus mConnectionStatus;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

3.1.42. WDSGetDataSessionDuration

Description	Queries the duration of the current call
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WDSGetDataSessionDuration()
	struct sWDSGetDataSessionDurationResponse_Duration
	{
	UINT64 mDataSessionDuration;
	};
	Optional TLVs
	// Structure to describe response TLV 0x10 for WDSGetDataSessionDuration()
	struct sWDSGetDataSessionDurationResponse_PreviousDuration
	{
	UINT64 mPreviousDataSessionDuration;
	};
	
	// Structure to describe response TLV 0x11 for WDSGetDataSessionDuration()

	struct sWDSGetDataSessionDurationResponse_ActiveDuration
	{
	UINT64 mDataSessionActiveDuration;
	};
	
	// Structure to describe response TLV 0x12 for WDSGetDataSessionDuration()
	struct sWDSGetDataSessionDurationResponse_PreviousActiveDuration
	{
	UINT64 mPreviousDataSessionActiveDuration;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_OUT_OF_CALL Call duration cannot be returned, since the call is not up
Constraint	

Example:

3.1.43. WDSGetDataBearerTechnology

Description	Queries the current data bearer technology
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WDSGetDataBearerTechnology()
	struct sWDSGetDataBearerTechnologyResponse_Technology
	{
	eQMIDataBearerTechnologies mDataBearerTechnology;
	};
	Optional TLVs
	// Structure to describe response TLV 0x10 for WDSGetDataBearerTechnology()
	struct sWDSGetDataBearerTechnologyResponse_LastCallTechnology
	{
	eQMIDataBearerTechnologies mDataBearerTechnology;
	};

Return Value	QMI_ERR_NONE No error in the request

	QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response QMI_ERR_OP_DEVICE_ UNSUPPORTED Operation is not supported by the device QMI_ERR_OUT_OF_CALL Data bearer is not returned since a call is not active
Constraint	

Example:

3.1.44. WDSGetChannelRates

Description	Queries the current bit rate of the packet data connection
Request Structure	

Response Structure	**Mandatory TLVs**
	// Structure to describe response TLV 0x01 for WDSGetChannelRates()
	struct sWDSGetChannelRatesResponse_ChannelRates
	{
	UINT32 mChannelTXRatebps;
	UINT32 mChannelRXRatebps;
	UINT32 mMaxChannelTXRatebps;
	UINT32 mMaxChannelRXRatebps;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point or the message was corrupted during transmission QMI_ERR_NO_MEMORY Device could not allocate memory to formulate a response
Constraint	

Example:

3.1.45. PDCReset

Description	Resets the PDC state variables of the requesting control point.
Request	

Structure	
Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response
Constraint	

Example:

3.1.46. PDCRegisterForIndications

Description	PDCRegisterForIndications

Request Structure	**Optional TLVs**
	// Structure to describe request TLV 0x10 for PDCRegisterForIndications()
	struct sPDCRegisterForIndicationsRequest_Config
	{
	INT8 mEnableReporting;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid
Constraint	
MSG ID	QMI_PDC_CONFIG_CHANGE_IND 0x0021

IND Structure	**Mandatory TLVs**
	// Structure to describe indication TLV 0x01 for PDC ConfigChangeIndication
	struct sPDCConfigChangeIndication_Config
	{
	eQMIPDCConfigurations mConfigType;
	UINT8 mConfigIDLength;

	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};

Example:

3.1.47. PDCGetSelectedConfig

Description	Gets the active configuration

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PDCGetSelectedConfig()
	struct sPDCGetSelectedConfigRequest_Type
	{
	eQMIPDCConfigurations mConfigType;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for PDCGetSelectedConfig()
	struct sPDCGetSelectedConfigRequest_Token
	{
	UINT32 mToken;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified
Constraint	
MSG ID	QMI_PDC_GET_SELECTED_CONFIG_IND 0x0022

IND Structure	**Mandatory TLVs**
	// Structure to describe indication TLV 0x01 for PDC GetSelectedConfigIndication
	struct sPDCGetSelectedConfigIndication_Error
	{

	eQMIErrors mQMIError;
	};
	Optional TLVs
	// Structure to describe indication TLV 0x10 for PDC GetSelectedConfigIndication
	struct sPDCGetSelectedConfigIndication_Token
	{
	UINT32 mToken;
	};
	
	// Structure to describe indication TLV 0x11 for PDC GetSelectedConfigIndication
	struct sPDCGetSelectedConfigIndication_ActiveID
	{
	UINT8 mConfigIDLength;
	
	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};
	
	// Structure to describe indication TLV 0x12 for PDC GetSelectedConfigIndication
	struct sPDCGetSelectedConfigIndication_PendingID
	{
	UINT8 mConfigIDLength;
	
	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};

Example:

3.1.48. PDCSetSelectedConfig

Description	Sets an available configuration for the device

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PDCSetSelectedConfig()
	struct sPDCSetSelectedConfigRequest_Config
	{
	eQMIPDCConfigurations mConfigType;
	UINT8 mConfigIDLength;

	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for PDCSetSelectedConfig()
	struct sPDCSetSelectedConfigRequest_Token
	{
	UINT32 mToken;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified QMI_ERR_ARG_TOO_LONG Specified argument size is too large
Constraint	
MSG ID	QMI_PDC_SET_SELECTED_CONFIG_IND 0x0023

IND Structure	// Structure to describe indication TLV 0x01 for PDC SetSelectedConfigIndication
	struct sPDCSetSelectedConfigIndication_Error
	{
	eQMIErrors mQMIError;
	};
	
	// Structure to describe indication TLV 0x10 for PDC SetSelectedConfigIndication
	struct sPDCSetSelectedConfigIndication_Token
	{
	UINT32 mToken;
	};

Example:

3.1.49. PDCListConfigs

Description	Lists the configurations currently available on the device

Request Structure	**Optional TLVs**
	// Structure to describe request TLV 0x10 for PDCListConfigs()
	struct sPDCListConfigsRequest_Token
	{
	UINT32 mToken;
	};
	
	// Structure to describe request TLV 0x11 for PDCListConfigs()
	struct sPDCListConfigsRequest_Type
	{
	eQMIPDCConfigurations mConfigType;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified QMI_ERR_ARG_TOO_LONG Specified argument size is too large
Constraint	
MSG ID	QMI_PDC_LIST_CONFIGS_IND 0x0024

IND Structure	// Structure to describe indication TLV 0x01 for PDC ListConfigsIndication
	struct sPDCListConfigsIndication_Error
	{
	eQMIErrors mQMIError;
	};
	
	// Structure to describe indication TLV 0x10 for PDC ListConfigsIndication
	struct sPDCListConfigsIndication_Token
	{

	UINT32 mToken;
	};
	
	// Structure to describe indication TLV 0x11 for PDC ListConfigsIndication
	struct sPDCListConfigsIndication_List
	{
	UINT8 mConfigListLength;
	
	struct sConfig
	{
	eQMIPDCConfigurations mConfigType;
	UINT8 mConfigIDLength;
	
	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};
	
	// This array must be the size specified by mConfigListLength
	// sConfig mConfigs[1];
	};

Example:

3.1.50. PDCDeleteConfig

Description	Deletes a configuration from the device

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PDCDeleteConfig()
	struct sPDCDeleteConfigRequest_Type
	{
	eQMIPDCConfigurations mConfigType;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for PDCDeleteConfig()
	struct sPDCDeleteConfigRequest_Token
	{
	UINT32 mToken;
	};

	// Structure to describe request TLV 0x11 for PDCDeleteConfig()
	struct sPDCDeleteConfigRequest_ID
	{
	UINT8 mConfigIDLength;
	
	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified QMI_ERR_ARG_TOO_LONG Specified argument size is too large
Constraint	
MSG ID	QMI_PDC_DELETE_CONFIG_IND 0x0025

IND Structure	**Mandatory TLVs**
	// Structure to describe indication TLV 0x01 for PDC DeleteConfigIndication
	struct sPDCDeleteConfigIndication_Error
	{
	eQMIErrors mQMIError;
	};
	Optional TLVs
	// Structure to describe indication TLV 0x10 for PDC DeleteConfigIndication
	struct sPDCDeleteConfigIndication_Token
	{
	UINT32 mToken;
	};

Example:

3.1.51. PDCLoadConfig

Description	Loads the specified configuration to device memory

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PDCLoadConfig()
	struct sPDCLoadConfigRequest_Frame1
	{
	eQMIPDCConfigurations mConfigType;
	UINT8 mConfigIDLength;
	
	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};
	
	struct sPDCLoadConfigRequest_Frame2
	{
	UINT32 mTotalSize;
	UINT16 mFrameSize;
	
	// This array must be the size specified by mFrameSize
	// UINT8 mFrame[1];
	};
	
	struct sPDCLoadConfigRequest_Frame
	{
	sPDCLoadConfigRequest_Frame1 mPDCLoadConfigRequest_Frame1;
	sPDCLoadConfigRequest_Frame2 mPDCLoadConfigRequest_Frame2;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for PDCLoadConfig()
	struct sPDCLoadConfigRequest_Token
	{
	UINT32 mToken;
	};

Response Structure	**Optional TLVs** // Structure to describe response TLV 0x10 for PDCLoadConfig() struct sPDCLoadConfigResponse_Reset

	{
	INT8 mFrameReset;
	};

Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified QMI_ERR_ARG_TOO_LONG Specified argument size is too large QMI_ERR_DEVICE_IN_USE Another control point is using this request
Constraint	
MSG ID	QMI_PDC_LOAD_CONFIG_IND 0x0026

IND Structure	// Structure to describe indication TLV 0x01 for PDC LoadConfigIndication
	struct sPDCLoadConfigIndication_Error
	{
	eQMIErrors mQMIError;
	};
	
	// Structure to describe indication TLV 0x10 for PDC LoadConfigIndication
	struct sPDCLoadConfigIndication_Token
	{
	UINT32 mToken;
	};
	
	// Structure to describe indication TLV 0x11 for PDC LoadConfigIndication
	struct sPDCLoadConfigIndication_Received
	{
	UINT32 mReceivedSize;
	};
	
	// Structure to describe indication TLV 0x12 for PDC LoadConfigIndication
	struct sPDCLoadConfigIndication_Remaining
	{
	UINT32 mRemainingSize;

	};
	
	// Structure to describe indication TLV 0x13 for PDC LoadConfigIndication
	struct sPDCLoadConfigIndication_Reset
	{
	INT8 mFrameReset;
	};

Example:

3.1.52. PDCActivateConfig

Description	Activates a pending configuration for the component

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PDCActivateConfig()
	struct sPDCActivateConfigRequest_Type
	{
	eQMIPDCConfigurations mConfigType;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for PDCActivateConfig()
	struct sPDCActivateConfigRequest_Token
	{
	UINT32 mToken;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified QMI_ERR_ARG_TOO_LONG Specified argument size is too large
Constraint	
MSG ID	QMI_PDC_ACTIVATE_CONFIG_IND 0x0027

IND Structure	**Mandatory TLVs**
	// Structure to describe indication TLV 0x01 for PDC ActivateConfigIndication
	struct sPDCActivateConfigIndication_Error
	{
	eQMIErrors mQMIError;
	};
	Optional TLVs
	// Structure to describe indication TLV 0x10 for PDC ActivateConfigIndication
	struct sPDCActivateConfigIndication_Token
	{
	UINT32 mToken;
	};

Example:

3.1.53. PDCGetConfigInfo

Description	Queries additional information for a configuration.

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PDCGetConfigInfo()
	struct sPDCGetConfigInfoRequest_Config
	{
	eQMIPDCConfigurations mConfigType;
	UINT8 mConfigIDLength;
	
	// This array must be the size specified by mConfigIDLength
	// UINT8 mConfigID[1];
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for PDCGetConfigInfo()
	struct sPDCGetConfigInfoRequest_Token
	{
	UINT32 mToken;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing

	QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified QMI_ERR_ARG_TOO_LONG Specified argument size is too large
Constraint	
MSG ID	QMI_PDC_GET_CONFIG_INFO_IND 0x0028

IND Structure	**Mandatory TLVs**
	// Structure to describe indication TLV 0x01 for PDC GetConfigInfoIndication
	struct sPDCGetConfigInfoIndication_Error
	{
	eQMIErrors mQMIError;
	};
	Optional TLVs
	// Structure to describe indication TLV 0x10 for PDC GetConfigInfoIndication
	struct sPDCGetConfigInfoIndication_Token
	{
	UINT32 mToken;
	};
	
	// Structure to describe indication TLV 0x11 for PDC GetConfigInfoIndication
	struct sPDCGetConfigInfoIndication_Size
	{
	UINT32 mTotalSize;
	};
	
	// Structure to describe indication TLV 0x12 for PDC GetConfigInfoIndication
	struct sPDCGetConfigInfoIndication_Description
	{
	UINT8 mDescriptionLength;
	
	// This array must be the size specified by mDescriptionLength
	// char mDescription[1];
	};

Example:

3.1.54. PDCGetConfigLimits

Description	Queries the maximum and current sizes for each configuration memory store

Request Structure	**Mandatory TLVs**
	// Structure to describe request TLV 0x01 for PDCGetConfigLimits()
	struct sPDCGetConfigLimitsRequest_Type
	{
	eQMIPDCConfigurations mConfigType;
	};
	Optional TLVs
	// Structure to describe request TLV 0x10 for PDCGetConfigLimits()
	struct sPDCGetConfigLimitsRequest_Token
	{
	UINT32 mToken;
	};

Response Structure	
Return Value	QMI_ERR_NONE No error in the request QMI_ERR_INTERNAL Unexpected error occurred during processing QMI_ERR_MALFORMED_MSG Message was not formulated correctly by the control point, or the message was corrupted during transmission QMI_ERR_NO_MEMORY Service could not allocate memory to formulate a response QMI_ERR_NOT_SUPPORTED Operation is not supported QMI_ERR_INVALID_ARG Specified parameter is invalid QMI_ERR_MISSING_ARG Required TLV is not specified
Constraint	
MSG ID	QMI_PDC_GET_CONFIG_LIMITS_IND 0x0029

IND Structure	**Mandatory TLVs**
	// Structure to describe indication TLV 0x01 for PDC GetConfigLimitsIndication
	struct sPDCGetConfigLimitsIndication_Error
	{
	eQMIErrors mQMIError;
	};
	Optional TLVs
	// Structure to describe indication TLV 0x10 for PDC GetConfigLimitsIndication
	struct sPDCGetConfigLimitsIndication_Token

```
{
    UINT32 mToken;
};

// Structure to describe indication TLV 0x11 for PDC GetConfigLimitsIndication
struct sPDCGetConfigLimitsIndication_MaximumSize
{
    UINT64 mMaximumSize;
};

// Structure to describe indication TLV 0x12 for PDC GetConfigLimitsIndication
struct sPDCGetConfigLimitsIndication_CurrentSize
{
    UINT64 mCurrentSize;
};
```

Example:

3.2. Function Prototypes

NULL

3.3. Complex types descripton

NULL

3.4. Constants

3.4.1. QMI service type values

QMI_CTL	0x00
QMI_WDS	0x01
QMI_DMS	0x02
QMI_NAS	0x03
QMI_QOS	0x04
QMI_WMS	0x05
QMI_PDS	0x06
QMI_AUTH	0x07
QMI_AT	0x08
QMI_VOICE	0x09
QMI_CAT	0x0A
QMI_UIM	0x0B
QMI_PBM	0x0C
QMI_QCHAT	0x0D
QMI_RMTFS	0x0E

QMI_TEST	0x0F
QMI_LOC	0x10
QMI_PDC	0x24

3.4.2. QMI error code

QMI_ERR_NONE	0x0000
QMI_ERR_MALFORMED_MSG	0x0001
QMI_ERR_NO_MEMORY	0x0002
QMI_ERR_INTERNAL	0x0003
QMI_ERR_ABORTED	0x0004
QMI_ERR_CLIENT_IDS_EXHAUSTED	0x0005
QMI_ERR_UNABORTABLE_TRANSACTION	0x0006
QMI_ERR_INVALID_CLIENT_ID	0x0007
QMI_ERR_NO_THRESHOLDS	0x0008
QMI_ERR_INVALID_HANDLE	0x0009
QMI_ERR_INVALID_PROFILE	0x000A
QMI_ERR_INVALID_PINID	0x000B
QMI_ERR_INCORRECT_PIN	0x000C
QMI_ERR_NO_NETWORK_FOUND	0x000D
QMI_ERR_CALL_FAILED	0x000E
QMI_ERR_OUT_OF_CALL	0x000F
QMI_ERR_NOT_PROVISIONED	0x0010
QMI_ERR_MISSING_ARG	0x0011
QMI_ERR_ARG_TOO_LONG	0x0013
QMI_ERR_INVALID_TX_ID	0x0016
QMI_ERR_DEVICE_IN_USE	0x0017
QMI_ERR_OP_NETWORK_UNSUPPORTED	0x0018
QMI_ERR_OP_DEVICE_UNSUPPORTED	0x0019
QMI_ERR_NO_EFFECT	0x001A
QMI_ERR_NO_FREE_PROFILE	0x001B
QMI_ERR_INVALID_PDP_TYPE	0x001C
QMI_ERR_INVALID_TECH_PREF	0x001D
QMI_ERR_INVALID_PROFILE_TYPE	0x001E
QMI_ERR_INVALID_SERVICE_TYPE	0x001F
QMI_ERR_INVALID_REGISTER_ACTION	0x0020
QMI_ERR_INVALID_PS_ATTACH_ACTION	0x0021
QMI_ERR_AUTHENTICATION_FAILED	0x0022
QMI_ERR_PIN_BLOCKED	0x0023
QMI_ERR_PIN_PERM_BLOCKED	0x0024
QMI_ERR_SIM_NOT_INITIALIZED	0x0025
QMI_ERR_MAX_QOS_REQUESTS_IN_USE	0x0026
QMI_ERR_INCORRECT_FLOW_FILTER	0x0027
QMI_ERR_NETWORK_QOS_UNAWARE	0x0028
QMI_ERR_INVALID_QOS_ID/QMI_ERR_INVALID_ID	0x0029
QMI_ERR_REQUESTED_NUM_UNSUPPORTED	0x002A
QMI_ERR_INTERFACE_NOT_FOUND	0x002B
QMI_ERR_FLOW_SUSPENDED	0x002C
QMI_ERR_INVALID_DATA_FORMAT	0x002D
QMI_ERR_GENERAL	0x002E
QMI_ERR_UNKNOWN	0x002F
QMI_ERR_INVALID_ARG	0x0030
QMI_ERR_INVALID_INDEX	0x0031

QMI_ERR_NO_ENTRY	0x0032
QMI_ERR_DEVICE_STORAGE_FULL	0x0033
QMI_ERR_DEVICE_NOT_READY	0x0034
QMI_ERR_NETWORK_NOT_READY	0x0035
QMI_ERR_CAUSE_CODE	0x0036
QMI_ERR_MESSAGE_NOT_SENT	0x0037
QMI_ERR_MESSAGE_DELIVERY_FAILURE	0x0038
QMI_ERR_INVALID_MESSAGE_ID	0x0039
QMI_ERR_ENCODING	0x003A
QMI_ERR_AUTHENTICATION_LOCK	0x003B
QMI_ERR_INVALID_TRANSITION	0x003C
QMI_ERR_NOT_A_MCAST_IFACE	0x003D
QMI_ERR_MAX_MCAST_REQUESTS_IN_USE	0x003E
QMI_ERR_INVALID_MCAST_HANDLE	0x003F
QMI_ERR_INVALID_IP_FAMILY_PREF	0x0040
QMI_ERR_SESSION_INACTIVE	0x0041
QMI_ERR_SESSION_INVALID	0x0042
QMI_ERR_SESSION_OWNERSHIP	0x0043
QMI_ERR_INSUFFICIENT_RESOURCES	0x0044
QMI_ERR_DISABLED	0x0045
QMI_ERR_INVALID_OPERATION	0x0046
QMI_ERR_INVALID_QMI_CMD	0x0047
QMI_ERR_TPDU_TYPE	0x0048
QMI_ERR_SMSC_ADDR	0x0049
QMI_ERR_INFO_UNAVAILABLE	0x004A
QMI_ERR_SEGMENT_TOO_LONG	0x004B
QMI_ERR_SEGMENT_ORDER	0x004C
QMI_ERR_BUNDLING_NOT_SUPPORTED	0x004D
QMI_ERR_OP_PARTIAL_FAILURE	0x004E
QMI_ERR_POLICY_MISMATCH	0x004F
QMI_ERR_SIM_FILE_NOT_FOUND	0x0050
QMI_ERR_EXTENDED_INTERNAL	0x0051
QMI_ERR_ACCESS_DENIED	0x0052
QMI_ERR_HARDWARE_RESTRICTED	0x0053
QMI_ERR_ACK_NOT_SENT	0x0054
QMI_ERR_INJECT_TIMEOUT	0x0055
QMI_ERR_INCOMPATIBLE_STATE	0x005A
QMI_ERR_FDN_RESTRICT	0x005B
QMI_ERR_SUPS_FAILURE_CAUSE	0x005C
QMI_ERR_NO_RADIO	0x005D
QMI_ERR_NOT_SUPPORTED	0x005E
QMI_ERR_NO_SUBSCRIPTION	0x005F
QMI_ERR_CARD_CALL_CONTROL_FAILED	0x0060
QMI_ERR_NETWORK_ABORTED	0x0061
QMI_ERR_MSG_BLOCKED	0x0062
QMI_ERR_INVALID_SESSION_TYPE	0x0064
QMI_ERR_INVALID_PB_TYPE	0x0065
QMI_ERR_NO_SIM	0x0066
QMI_ERR_PB_NOT_READY	0x0067
QMI_ERR_PIN_RESTRICTION	0x0068
QMI_ERR_PIN2_RESTRICTION	0x0069
QMI_ERR_PUK_RESTRICTION	0x006A
QMI_ERR_PUK2_RESTRICTION	0x006B

QMI_ERR_PB_ACCESS_RESTRICTED	0x006C
QMI_ERR_PB_DELETE_IN_PROG	0x006D
QMI_ERR_PB_TEXT_TOO_LONG	0x006E
QMI_ERR_PB_NUMBER_TOO_LONG	0x006F
QMI_ERR_PB_HIDDEN_KEY_RESTRICTION	0x0070
QMI_ERR_PB_NOT_AVAILABLE	0x0071

3.5. Enumerations

NULL