ACKNOWLEDGEMENT

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

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

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

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

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

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

Local/Global Phonebook Access Through QMI-PBM Interface

Application Note

80-N3062-1 A

September 21, 2010

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

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

Contents

Figures

Tables

Revision history

Revision	Date	Description
A	Sep 2010	Initial release

1 Introduction

1.1 Purpose

This document discusses an option that allows local/global phonebooks of the UIM card to be accessed through the QMI-PBM interface.

1.2 Scope

This document is applicable for all customers who would like to access phonebook information from the UIM card using the QMI interface instead of the RPC-based interface.

1.3 Conventions

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

1.4 References

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

Table 1-1 Reference documents and standards

Ref.	Document	
Qualcomm		
Q1	*Application Note: Software Glossary for Customers*	CL93-V3077-1
Q2	*QMI Phonebook Manager Service (QMI_PBM)*	80-VB816-15

1.5 Technical assistance

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

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

1.6 Acronyms

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

2 Software Update Description

2.1 Problem with read

Currently, QCRIL on the apps processor side uses the MMGSDI RPC interface for importing card contacts. A side effect of reading ADN contacts through the MMGSDI interface on power-up is that the total initialization time of the entire ADN phonebook is long. The PBM module on top of MMGSDI on the modem processor is fetching ADN records simultaneously. (PBM caches the card contacts to facilitate ATCoP residing on the modem processor side to perform phonebook read/write operations.) As these two activities (QCRIL and PBM fetch) occur independently, there is an interleaving problem at UIM because of different file select and read operations at the I/O level.

2.2 Problem with write

If the same MMGSDI interface is used for writing contacts, the PBM module goes out of sync, so that stale information is returned for phonebook-related AT commands. This is because MMGSDI has no intimation mechanism to its clients for updates made to EFs of PBM interest unless they are refreshed over-the-air. If all updates occur via PBM, the cache will be up to date.

2.3 Update

Two alternative options exist to resolve this issue:

- Use the PBM-RPC-based approach.
- Use a simplified QMI-PBM interface.

This document highlights the second option. For this option, all the associated contact data, e.g., corresponding number, name, email, must be available for a write request. See Section 3.5 of [Q2] for more specific details.

Contact Qualcomm CDMA Technologies for sample code (see Section 1.5).

2.4 Side effects of the change

No side effect from this change is anticipated.

3 QCCI APIs

The QCCI interface provides a distinct set of APIs for communicating to the modem processor. The use of these APIs can be better understood when the sample code is provided via a patch release.

The QMI APIs can be divided into the following broad categories:

- Connection APIs
- Message-sending APIs
- Release connection APIs
- Encode/decode APIs

The QMI framework should be initialized through the qmi_init() per process. The qmi_release() function should be called after all the QMI work is done.

3.1 Connection APIs

qmi_client_init

This function is used for initializing a connection to a service. It sets the user_handle and returns QMI_NO_ERR if it is successful in regard to error codes.

```
extern qmi_client_error_type
qmi_client_init
(
  const char                              *conn_id,
  qmi_idl_service_object_type             service_obj,
  qmi_client_ind_cb                       ind_cb,
  void                                    *ind_cb_data,
  qmi_client_type                         *user_handle
);
```

3.2 Message-sending APIs – Asynchronous

3.2.1 qmi_client_send_raw_msg_async

This API sends an asynchronous QMI service message on the specified connection. It returns QMI_NO_ERR and sets the transaction handle if the function is successful with regard to error codes. The QMI framework calls the user-defined callback resp_cb as and when a response is received from the modem side.

The transaction handle can be used to cancel the transaction after sending the message. The user of this API is responsible for decoding the raw data and for memory management of the request and response buffers.

```
extern qmi_client_error_type
qmi_client_send_raw_msg_async
(
  qmi_client_type                user_handle,
  unsigned long                  msg_id,
  void                           *req_buf,
  int                            req_buf_len,
  void                           *resp_buf,
  int                            resp_buf_len,
  qmi_client_recv_raw_msg_async_cb  resp_cb,
  void                           *resp_cb_data,
  qmi_txn_handle                 *txn_handle
);
```

3.2.2 qmi_client_send_msg_async

This API sends an asynchronous QMI service message on the specified connection. It returns QMI_NO_ERR, and sets the transaction handle if the function is successful. Otherwise, it returns an error code.

The QMI framework decodes the response received and calls the resp_cb callback defined by the user with decoded data in the response structure. This API must be preferred over the qmi_client_send_raw_msg_async API, as the user does not have to encode, decode, and manage the memory for the response and receive buffer.

NOTE: This is imperative to avoid processing that takes a long time in the callback, which might stall the entire system. Qualcomm strongly advises against sending async or sync messages from within the callback.

```
qmi_client_error_type
qmi_client_send_msg_async
(
    qmi_client_type                 user_handle,
    unsigned long                   msg_id,
    void                            *req_c_struct,
    int                             req_c_struct_len,
    void                            *resp_c_struct,
    int                             resp_c_struct_len,
    qmi_client_recv_msg_async_cb    resp_cb,
    void                            *resp_cb_data,
    qmi_txn_handle                  *txn_handle
);
```

3.2.3 qmi_client_delete_async_txn

This function can be used to cancel an async transaction. The async_txn_handle is returned by the qmi_client_send_msg_async function. It returns QMI_NO_ERR if successful. Otherwise, it returns an error code. Users should be aware of the potential race condition in which an asynchronous response may be in the process of being handled by the users_rsp_cb callback until this routine returns.

```
extern qmi_client_error_type
qmi_client_delete_async_txn
(
  qmi_client_type     user_handle,
  qmi_txn_handle      async_txn_handle
);
```

3.3 Message-sending APIs – Synchronous

3.3.1 qmi_client_send_raw_msg_sync

This API sends a synchronous QMI service message on the specified connection. It returns QMI_NO_ERR, and sets resp_buf_recv_len if the function is successful. Otherwise, it returns an error code.

This API expects the user to encode the message before sending and to decode the message after receiving. The memory management of the buffer is also left to the user. Since the API sends a synchronous message that it blocks while waiting for the response, the user can time out by using the timeout specified in milliseconds.

```
extern qmi_client_error_type
qmi_client_send_raw_msg_sync
(
  qmi_client_type              user_handle,
  unsigned long                msg_id,
  void                         *req_buf,
  int                          req_buf_len,
  void                         *resp_buf,
  int                          resp_buf_len,
  int                          *resp_buf_recv_len,
  int                          timeout_msecs
);
```

3.3.2 qmi_client_send_msg_sync

This API sends a synchronous QMI service message on the specified connection. It returns QMI_NO_ERR if the function is successful. Otherwise, it returns an error code.

This function provides the encoding/decoding functionality, and the user receives the decoded data in the provided response structure.

```
extern qmi_client_error_type
qmi_client_send_msg_sync
(
    qmi_client_type        user_handle,
    int                        msg_id,
    void                    *req_c_struct,
    int                        req_c_struct_len,
    void                       *resp_c_struct,
    int                        resp_c_struct_len,
    int                        timeout_msecs
);
```

3.4 Release connection APIs

This function releases the connection. It returns QMI_NO_ERROR if the function is successful. Otherwise, it returns an error code.

```
extern qmi_client_error_type
qmi_client_release
(
  qmi_client_type     user_handle
);
```

3.5 Encode/decode APIs

3.5.1 qmi_client_message_encode

This API encodes the body (TLV) of a QMI message from the C data structure to the wire format. If successful, it returns QMI_NO_ERR and sets the number of bytes encoded in dst_encoded_len. Otherwise, it returns an error code.

```
extern qmi_client_error_type
qmi_client_message_encode
(
    qmi_client_type                 user_handle,
    qmi_idl_type_of_message_type    req_resp_ind,
    int                             message_id,
    const void                      *p_src,
    int                             src_len,
    void                            *p_dst,
    int                             dst_len,
    int                             *dst_encoded_len
);
```

3.5.2 qmi_client_message_decode

This API decodes the body (TLV) of a QMI message body from the wire format to the C structure.

```
extern qmi_client_error_type
qmi_client_message_decode
(
    qmi_client_type                 user_handle,
    qmi_idl_type_of_message_type    req_resp_ind,
    int                             message_id,
    const void                      *p_src,
    int                             src_len,
    void                            *p_dst,
    int                             dst_len
);
```

3.5.3 qmi_client_get_service_version

This API provides major/minor version information pertaining to a service object and a connection ID. For standard QMI error codes, see qmi.h. The QCCI APIs are defined in qmi_client.h.

```
qmi_client_error_type
qmi_client_get_service_version
(
    const char                    *conn_id,
    qmi_idl_service_object_type   service_obj,
    qmi_service_version_info      *service_version_info
);
```

3.6 Call flow between apps and modem processors

Figure 3-1 illustrates message flow between the apps processor and the modem processor (initialization and reading ADN records).

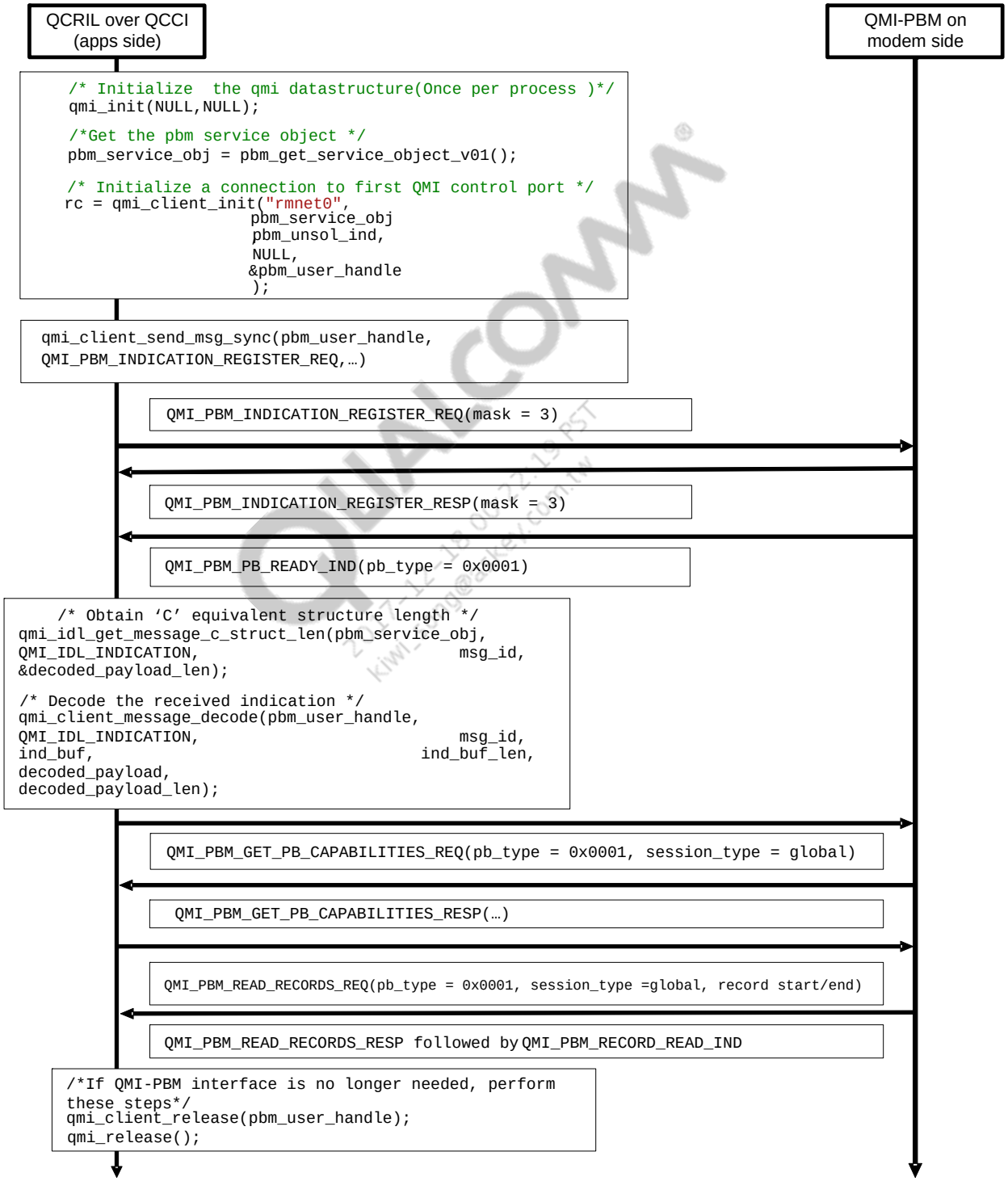

Figure 3-1 Message flow

NOTE: Customers must add the code shown in boxes in Figure 3-1 (apps side).

3.7 Usage notes

The source code has been added to qmi_simple_ril_suite.c, which is located under vendor/qcom/proprietary/qmi/tests.

To test QMI PBM functionality, pass pbm as the argument when running the qmi_simple_ril_test executable, i.e., do the following in adb shell (after getting to the appropriate directory on the device and changing permissions, if needed):

```
./qmi_simple_ril_test pbm
```

The binaries that have changed due to the addition of this code are qmi_simple_ril_test and qmilibservices.so.

NOTE: The standalone test code employing the QMI-PBM interface is supplied for reference only.

NOTE: **QCRIL changes are to be performed by the customer** – It is expected that QMI_PBM_PB_READY_IND is first tapped before PBM capabilities are queried and ADN records are fetched. Near the end of qcril_init() and qcril_release() functions, respective QCCI initialization and release code snippets may be added, as highlighted in the call flow. Based on the sample code, suitable modifications should be made to qcril_mmgsdi_request_sim_io(), if it is an ADN request, to perform a QMI read or write request. Fusion QCRIL uses a request list mechanism to correlate the incoming response/indication from the modem processor. The customer must do some code manipulation around the fact that the responses/indications from the modem for QMI-PBM requests are different from RPC events.