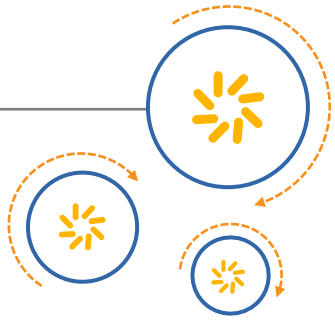
Qualcomm Technologies, Inc.



Sensors Execution Environment Client API

Reference

80-P9301-36 A

June 26, 2017

Qualcomm
2019-04-14 23:53:43 PDT
zk_sw@wingtech.com

Confidential and Proprietary – Qualcomm Technologies, Inc.

NO PUBLIC DISCLOSURE PERMITTED: Please report postings of this document on public servers or websites to: DocCtrlAgent@qualcomm.com.

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

Not to be used, copied, reproduced, or modified in whole or in part, nor its contents revealed in any manner to others without the express written permission of Qualcomm Technologies, Inc.

Qualcomm Hexagon is a product of Qualcomm Technologies, Inc. Other Qualcomm products referenced herein are products of Qualcomm Technologies, Inc. or its subsidiaries.

Qualcomm and Hexagon are trademarks of Qualcomm Incorporated, registered in the United States and other countries. Other product and brand names may be trademarks or registered trademarks of their respective owners.

This technical data may be subject to U.S. and international export, re-export, or transfer (“export”) laws. Diversion contrary to U.S. and international law is strictly prohibited.

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

© 2017 Qualcomm Technologies, Inc. All rights reserved.

Revision history

Revision	Date	Description
A	June 2017	Initial release

Qualcomm
2019-04-14 23:53:43 PDT
zk_sw@wingtech.com

Contents

Qualcomm 2019-04-14 23:33:43 PDT ak_sw@wimaxtech.com

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

1 Introduction

1.1 Purpose

This document describes the functions of the sensor clients with the Sensors Execution Environment (SEE). This document complements the example client code located in the Linux Android build, which provides examples to most of the referenced messages and procedures. The build file is in the <build>/vendor/qcom/proprietary/sensors-see/test/sns_client_example/src/sns_client_example.c file.

1.2 Conventions

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

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

1.3 Technical assistance

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

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

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

2 Client API

Sensor clients access the SEE, which resides on the Snapdragon Sensor Core (SSC), via the sensors Qualcomm Messaging Interface (QMI) service. For additional information on the QMI Common Client Interface (QCCI), see *QMI Client API Interface Specification* (80-N1123-1). All clients on all processors and targets use the same interface.

2.1 Connections

Sensor clients must first open a QCCI connection to send requests to the SEE. As the client may be created prior to SSC initialization, Qualcomm recommends that clients register for a callback when the sensors service becomes available using `qmi_client_notifier_init()`. Upon receipt of the callback, clients open a connection to the sensors service using `qmi_client_init_instance()`. This function is a variant of `qmi_client_init()`, where the client must also indicate to which instance of the service they wish to connect. The sensors service may be registered on multiple processors, other than just the SSC. For most clients, the SSC-based instance ID of 0 is appropriate.

The typical client also registers for error callbacks via `qmi_client_register_error_cb()`. These callbacks occur when the service becomes unavailable, typically due to a subsystem restart (SSR) or protection domain (PD) restart. If the SEE restarts, all client states are lost, and the client must repeat its initialization and resend any active request messages.

2.2 Messages

Sensors use QMI as the transportation mechanism for messages. QCCI/QMI Common Service Interface (QCSI) supports only the transport of IDL-defined messages. Therefore, Qualcomm defined an sns_client_api_v01.idl file. The following flow is observed for all requests.

1. Sends a `sns_client_req_msg` request message via `qmi_client_send_msg_async()`
   - The sole field of this message is *payload*, which is an opaque byte array. This field is populated with the protocol buffer-encoded `sns_client_request_msg`.
2. Receives a `sns_client_resp_msg` response message
   a. The client manager sends this response message immediately upon receipt of the request.
   b. A minimal amount of processing is performed on the request. The client manager determines if the `sns_client_request_msg` is properly encoded and whether the destination sensor unique identifier (SUID) is available.
   c. The client manager returns the client ID for the QMI connection. During the connection creation process, the ID is assigned and reserved for the life of the connection.
   d. The result (b) and client ID (c) are returned in the response message.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

e. In lieu of a response message, QCCI may instead specify a non-zero `transp_err` value. The integer value refers to the specific error that occurred. Several common errors are:

- (-41) to (-52) – QMI encoding error; client-provided message is improperly formed
- QMI_SERVICE_ERR – Service is no longer available

3. Receives one or more indications of `sns_client_report_ind_msg` type.

a. The indication messages are received within the indication callback function, as specified during the connection creation.

b. Each indication message contains a unique client ID.

c. The indication message contains a single *payload* field. Similar to the request message, it is intended to contain a protocol buffer-encoded message of `sns_client_event_msg` type.

d. An indication message may contain one or more logical sensor samples that are all encoded in the single `sns_client_event_msg`.

Qualcomm
2019-04-14 23:53:43 PDT
zk_sw@wingtech.com

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

3 Protocol buffers

QMI request and indication messages are opaque memory buffers, inside of which protocol buffer-encoded messages are placed. Clients can generate these messages in several different programming languages and then copy the encoded byte stream into the `sns_client_request_msg`. Similarly, clients copy the encoded message from within `sns_client_report_ind_msg` and decode it separately.

For more information, see the software architecture figures in *SDM845 Sensors Overview* (80-P9301-34).

For more information on protocol buffers, see https://developers.google.com/protocol-buffers/ and https://github.com/nanopb/nanopb.

Qualcomm
2019-04-14 23:53:43 PDT
zk_sw@wingtech.com

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

4 Client manager

The sensors QMI client manager resides on the SSC and handles all QMI communication. It is responsible for translating incoming QMI requests into request messages understood by the SEE and translating event messages received from the SEE into outgoing QMI indications. The client manager is also responsible for guaranteeing all batching options specified by the client.

For additional information, see the sns_client.proto file.

4.1 SUID

The SUID is an integer that uniquely identifies a single sensor available from the SEE. Clients act as though this ID is generated randomly upon boot-up and make no assumptions regarding repeatability, nor gleam any information from the number itself. If there are two gravity algorithms available from the SEE, each has its own SUID. If the hardware for a physical sensor model is present multiple times on the device, each has its own SUID. For a complete list of sensors available on the system, use the SUID Lookup Sensor to initiate a query.

4.2 Client request

All incoming requests to the client manager use `sns_client_request_msg` as the outermost protocol buffer message. This message contains the following fields:

- SUID – Serves as the destination address of the request message. All request messages must be sent to a valid SUID for processing; otherwise, the client manager rejects them.
- Message ID – Numerical identifier for the encoded request contained in `sns_std_request::payload`. For example, a sensor may support an *enable streaming* request message, as well as an *initiate data flush* request. The formats of these two messages are different, and this field indicates to the destination sensor how to interpret the request. Message IDs are always unique among all messages supported by a sensor; however, two different sensors may use a particular ID for different message types.
- Request – Everything from here on is passed directly to the sensor for processing. The `sns_std_request::payload` field contains the sensor-specific configuration and corresponds to the message ID.
- `suspend_config::client` – Processor on which the client resides. If any client on a processor requests a flush, all data batched for all clients on that processor are sent at once. This is done for power optimization purposes.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

- suspend_config::wakeup
  - SNS_STD_DELIVERY_WAKEUP – Sends events whenever they become available (at sample rate or batch period). If a batch_period larger than system capacity is requested, all data is sent upon capacity exhaustion. With this option, the flush_period is effectively ignored, as unsent batched data does not have the opportunity to accrue in the buffer.
  - SNS_STD_DELIVERY_NO_WAKEUP – Sends events only when the client processor is not in suspend; otherwise, batches indefinitely. Data for clients that specified a batch_period of 0 is also batched here. Once the target processor exits suspend, all pending events are sent.

4.2.1 Batching

Within sns_client_request_msg:: sns_std_request, the client has the opportunity to specify how and when it receives its requested data.

- batching::batch_period – A client can assume a timer is registered for batch_period microseconds. All events generated since the last timer expiration are saved until the subsequent firing. This period is interpreted as a maximum period specified by the client. Events may be delivered to client at a faster rate (smaller batch period) in some concurrency scenarios. A client may send a flush request at any time to instruct the client manager to send all batched data. A batch period of 0 indicates that no batching occurs; batching is disabled by default.
- batching::flush_period – This field provides a hint to the client manager or physical sensor regarding how much historical data should be batched if for whatever reason data is not being sent to the client. In other words, it instructs the client manager not to store or return more than approximately batch_period microseconds of data upon a flush. The effective flush period may be smaller due to system memory constraints or larger in concurrency cases. This field is optional, and if not set, defaults to batch_period, that is, only a single batch of data is maintained.
  - If set to 0, it is a suggestion to the sensor that data need not ever be sent or batched.
- batching::flush_only – Sends events only upon a client-initiated flush. Otherwise, it continues batching until flush_period is reached (at which time, batching continues, but oldest data may be dropped).
- batching::max_batch – Power-optimized mode; it directs sensor drivers to configure their hardware to minimize Sensor Low Power Island (SLPI) wake-ups.

4.3 Client event

All outgoing indications use sns_client_event_msg as the outermost protocol buffer-encoded message, within the payload field of sns_client_report_ind_msg. This message contains several fields.

- SUID – Same as the SUID specified in the request message that prompted this event. If a client sends requests to multiple SUIDs on a single connection, this message only contains the events for one SUID; events from other SUIDs are delivered in separate indication messages.
- Message ID – Uniquely identifies the associated event message.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

- Timestamp – Timestamp of the sensor event, in QTimer clock ticks. A QTimer tick is defined as one cycle of the Qualcomm® Hexagon™ processor's 19.2 MHz QTimer clock. For most events, the timestamp is set by the sensor and refers to the time at which the physical sample was created in the sensor hardware.
- Payload – The sensor-specific event.

Qualcomm
2019-04-14 23:53:43 PDT
zk_sw@wingtech.com

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

5 Message payloads

The sensor-specific request or event is referred to in `sns_client_request_msg:: sns_std_request::payload` and `sns_client_event_msg:: sns_client_event::payload`. These fields contain a protocol buffer-encoded message with fields specific to that message ID, or may be empty and have no fields.

Clients use the .proto file associated with the sensor in which they communicate. Each sensor type has a corresponding .proto file. For example, sns_accel.proto describes how to enable an accelerometer stream. In addition, every sensor publishes its list of .proto files (see Chapter 6).

5.1 Data types

Each sensor advertises a data type attribute (see Chapter 6). Each type is associated with a unique set of *.proto files that make up the sensor-specific API for that sensor. All sensors of the same type must support a minimum set of request and event messages; they may define and use additional optional messages that are specific to that sensor implementation. Each sensor publishes a complete list of .proto files.

5.2 Standardized messages

Any client may send a set of Qualcomm-defined standardized messages to any sensor. These messages are defined in the sns_std.proto file.

- `SNS_STD_MSGID_SNS_STD_ATTR_REQ` – Queries a sensor for its list of attributes. Returns an event with ID `SNS_STD_MSGID_SNS_STD_ATTR_EVENT`, containing a list of all published attributes (see Chapter 6). Clients may also register for notification when a new or updated attribute is published.
- `SNS_CLIENT_MSGID_SNS_CLIENT_DISABLE_REQ` – Disables the active request for this sensor. For example, the client sends a `SNS_STD_SENSOR_MSGID_SNS_STD_SENSOR_CONFIG` message to the accelerometer sensor to enable streaming. Subsequently, sending a `DISABLE_REQ` cancels the streaming request, and accelerometer streaming ceases for this client.
- `SNS_STD_MSGID_SNS_STD_FLUSH_REQ` – Forces all batched data from this sensor to be immediately sent to the client. This command forces the applicable hardware and software buffers on the system to flush all data present.

  For example, sending this request to an accelerometer sensor causes the physical FIFO to be flushed, as well as any samples currently held by the client manager. If the flush request is to an algorithm, such as game rotation vector, which internally uses accelerometer and gyroscope data, both the accelerometer and gyroscope hardware FIFOs are flushed.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

5.3 Standardized sensor messages

The messages described in Section 4.2 are applicable to all sensors. In addition to those, Qualcomm defined a set of standardized messages recommended for use by sensor developers. These recommendations are ultimately optional, and developers may instead choose to define their own request and event messages

- `SNS_STD_SENSOR_MSGID_SNS_STD_SENSOR_CONFIG` – Request message that enables any streaming physical sensor (for example, accelerometer, gyroscope, magnetometer) and some algorithms (for example, rotation vector)
- `SNS_STD_SENSOR_MSGID_SNS_STD_SENSOR_EVENT` – Data sample produced by the sensor
- `SNS_STD_SENSOR_MSGID_SNS_STD_SENSOR_PHYSICAL_CONFIG_EVENT` – Sent by all physical sensors upon processing a client request; indicates what data stream clients should expect (for example, the rate at which the sensor produces samples)

For additional information, see the sns_std_sensor.proto file.

5.4 SUID lookup sensor

Clients may query the SUID lookup sensor for the list of SUIDs associated with a specific data type. The SUID lookup request specifies a data type, and the client receives all matching SUIDs. An empty data type string results in the receipt of the list of all SUIDs on the system.

For example, a client could specify “accel” as the data type and receives a list of all SUIDs whose sensors provide accelerometer data. The client may also specify whether to register for notifications when a new match occurs. This may be followed-up by an attribute request to determine which of the available accelerometer sensors are appropriate for this client.

The SUID lookup sensor has its own SUID, which is a constant published in its .proto file (it is unique among all other sensors in this way).

For additional information, see the sns_suid.proto file.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

6 Sensor attributes

Every sensor publishes a list of attributes, where each attribute is represented by a numerical identifier. These attributes provide information on the sensor capabilities and the allowed range of values it would accept as input. Several important attributes are:

- `SNS_STD_SENSOR_ATTRID_TYPE` – A character string representing the data type this sensor supports (for example, accel).
- `SNS_STD_SENSOR_ATTRID_API` – List of the .proto filenames used by this sensor; additional .proto dependencies may be specified as imports within this file. Used primarily for test automation.
- `SNS_STD_SENSOR_ATTRID_EVENT_SIZE` – The size in bytes of the data event (protocol buffer encoded) produced by this sensor; for physical and virtual sensors, this refers to the size of their sensor sample. Used by HAL for maximum batching capacity determination.
- `SNS_STD_ATTRID_BATCH_MEM` – The total amount of batching space available for this client, in bytes; all clients share this space. A client can determine approximately the maximum number of samples that can be stored of this sensor type by using this attribute in conjunction with `EVENT_SIZE`.

Qualcomm
2019-04-14 15:43:57
zk_sw@wingtech.com

7 Use-case limitations

Clients may send any number of requests to any number of sensors on a single QMI connection. However, only a single active stream per sensor is allowed. For example, a client sends an enable request to a SUID representing an accelerometer sensor and specifies a sample rate of 50 Hz. Soon after, the client expects to begin receiving indication messages containing accelerometer samples at or above the rate of 50 Hz. The client subsequently sends another enable request to the same SUID, this time specifying 10 Hz. Rather than receiving the 10 Hz and the 50 Hz data, the 50 Hz request is replaced with 10 Hz, and the client only receives accelerometer data at 10 Hz.

If several concurrent streams or requests to a sensor are required, those must be sent over separate QMI connections.

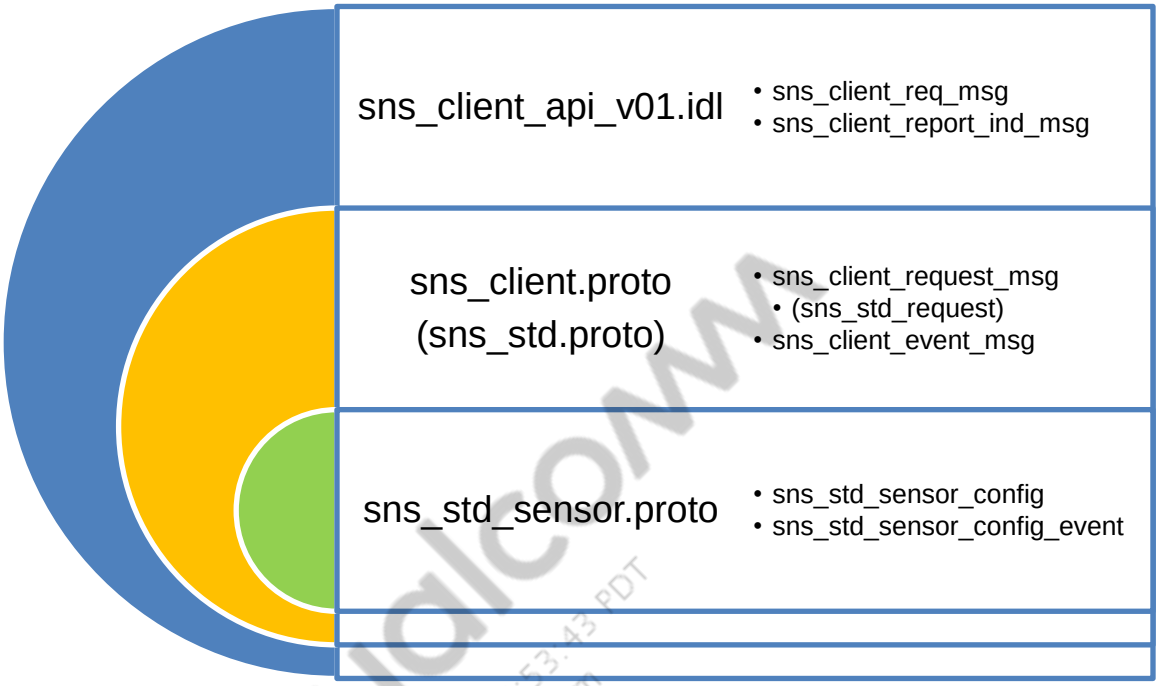
After a client sends an enable request to any sensor, it should expect a prompt event containing the configuration chosen by that sensor. For example, if a client sends a 50 Hz enable request to the accelerometer sensor, the accelerometer sensor typically must choose the next highest available output data rate (ODR) to program the hardware, for example, 60 Hz. The client receives a configuration event from the accelerometer sensor specifying a 60 Hz rate.

Similarly, on-change (also occasionally referred to as event-based) sensors always return an initial state event, soon after processing the enable request from a new client. This event contains the current state of the sensor.

Most physical sensor APIs only guarantee that the received rate is equal to or greater than the requested rate. Therefore, clients must be prepared to receive data potentially at a much faster rate than requested. Some physical sensor clients have strict requirements regarding their requested sampling rate. For these clients, Qualcomm recommends using the resampler sensor to achieve the desired sampling rate. This sensor performs filtering and interpolation on the physical sensor samples to achieve the requested output rate. For more information, see sns_resampler.proto file.

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION

The figure shows the layers and nested structure of the messages discussed in Section 2.2 and Chapter 4.



Qualcomm
2019-04-14 23:53:43 PDT
zk_sw@wingtech.com

A References

A.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
QMI Client API Interface Specification	80-N1123-1
SDM845 Sensors Overview	80 P9301-34
Resources	
Google, Inc.	https://developers.google.com/protocol-buffers/ https://github.com/nanopb/nanopb

A.2 Acronyms and terms

Acronym or term	Definition
ODR	Output data rate
PD	Protection domain
QCCI	QMI Common Client Interface
QCSI	QMI Common Service Interface
QMI	Qualcomm Messaging Interface
SEE	Sensors Execution Environment
SLPI	Sensor Low Power Island
SSC	Sensors Snapdragon Core
SSR	Subsystem restart
SUID	Sensor unique identifier

Qualcomm

MAY CONTAIN U.S. AND INTERNATIONAL EXPORT CONTROLLED INFORMATION