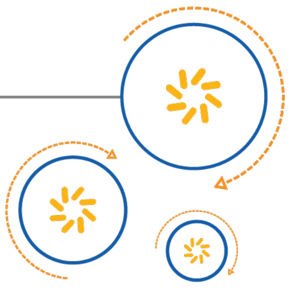
Qualcomm Technologies, Inc.



Unified Sensor Test Application (USTA)

User Guide

80-P9301-85 Rev. A

February 16, 2018

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

Revision history

Revision	Date	Description
A	February 2018	Initial release

Contents

1 Introduction

1.1 Purpose

This document describes how to use the Unified Sensor Test Application (USTA) for sensor streaming. USTA is part of current software releases and is enabled with default Qualcomm® builds provided for Android.

This document covers the USTA UI usage as well as command-line options.

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>*.

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

1.3 Technical assistance

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

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

2 Getting started

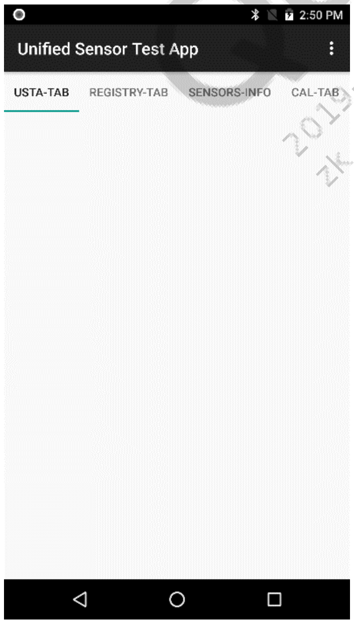
USTA dynamically supports all sensor types available from the Sensors Execution Environment (SEE) client API. USTA creates and sends requests to any sensor available with the SEE framework, parses all the events back to the client, and displays the events in the application UI and adb logcat.

The USTA source code is available in Android builds at `LINUX/android/vendor/qcom/proprietary/sensors-see/USTA`.

1. Run the following commands before starting USTA:

```
adb root
adb setenforce 0
adb reboot
```

2. Start the Unified Sensor Test App application. A truncated version of the title may appear on an MTP.



The default UI appears.

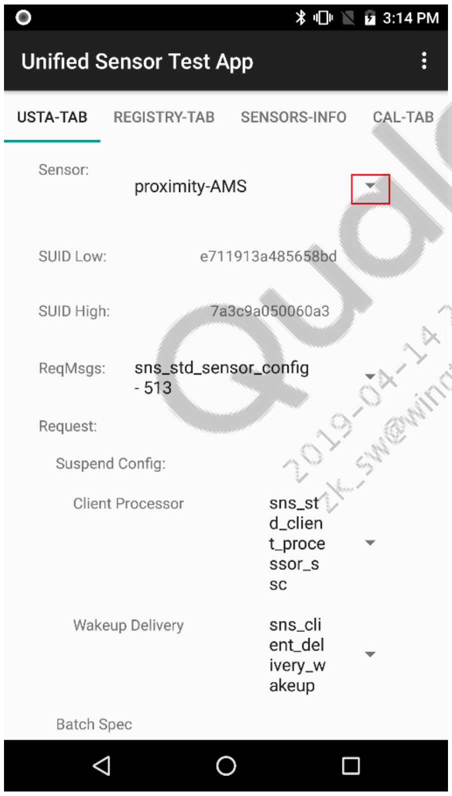
The default UI contains the following tabs:

- USTA-TAB – Sends a request to any sensor supported by the SEE framework
- REGISTRY-TAB – Reads and modifies registry information for various sensors
- SENSORS-INFO – Displays the attributes for each of the sensors identified by USTA

  By default, a sensor with a zero (0) handle displays its attributes. To view the attributes of a sensor, select the sensor from the available list and click the down arrow.
- CAL-TAB – Helps execute the user-cal for accelerometer via JNI

3 Using USTA UI to stream sensors

1. Start USTA. The default UI appears.
2. To view the sensors listed on USTA, click the **SENSORS-INFO** tab and then click the **USTA-TAB** tab.



3:14 PM

Unified Sensor Test App

Sensor: proximity-AMS

SUID Low: e711913a485658bd

SUID High: 7a3c9a050060a3

ReqMsgs: sns_std_sensor_config - 513

Request:

Suspend Config:

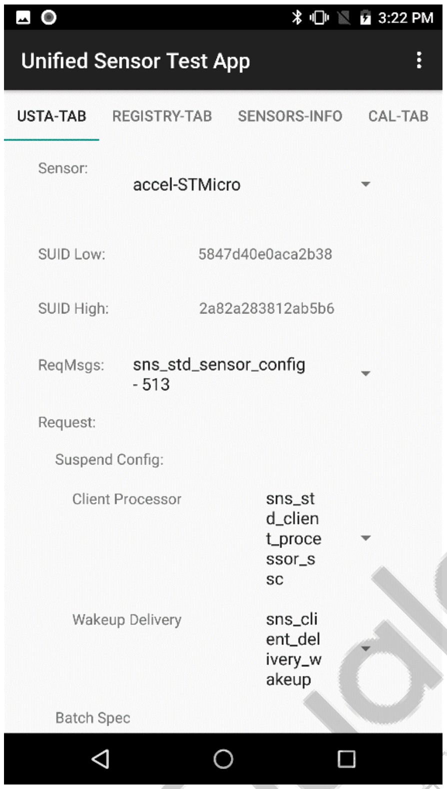
Client Processor sns_std_client_processor_ssc

Wakeup Delivery sns_client_delivery_wakeup

Batch Spec

The sensor name is concatenated with its vendor name. For all sensors, the name appears in front of Sensor field in a sensor_name-vendor_name format. The SUID for a selected sensor appears in the SUID Low and SUID High text fields. Other options are available as either drop-down lists, check boxes, or as fields in which information is entered, based on each sensors .proto file.

3. To stream a sensor using the USTA UI, select one of the sensors in the sensor list.



Sensor: accel-STMicro

SUID Low: 5847d40e0aca2b38

SUID High: 2a82a283812ab5b6

ReqMsgs: sns_std_sensor_config - 513

Request:

Suspend Config:

Client Processor sns_std_client_processor_ssc

Wakeup Delivery sns_client_delivery_wakeup

Batch Spec

Multiple message requests appear in ReqMsgs.

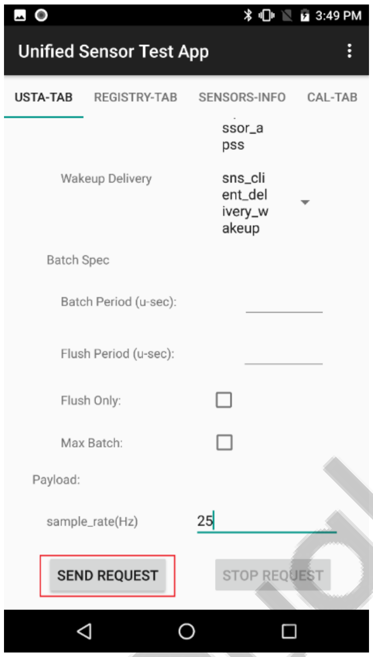
a. To view the payload fields, select one of the request messages in ReqMsgs, for example, **sns_std_sensor_config-513**.

   Payload fields vary according to the per-request message based on the .proto files. For most sensors, sns_std_sensor_config-513, sns_std_on_change_request-514, and sns_physical_sensor_test_config-515 are used for streaming. Refer to the related sensor .proto file before making a request.

b. In Request > Suspend Config, the Client Processor and Wakeup Delivery type must be selected. For streaming an accelerometer sensor to the application processor, select **sns_std_cleint_processor_apss** and then select **sns_client_delivery_wakeup** to wake up the application processor when the samples arrive.

c. For accelerometer, fields in Batch Spec are optional. When requesting a batching request, complete these fields. For example, if data is needed every 1 sec, set the batch period to **1000000** (microseconds).

d. In Payload, type the sampling rate and click **SEND REQUEST**.



ssor_a
pss

Wakeup Delivery: sns_client_delivery_wakeup

Batch Spec

Batch Period (u-sec):

Flush Period (u-sec):

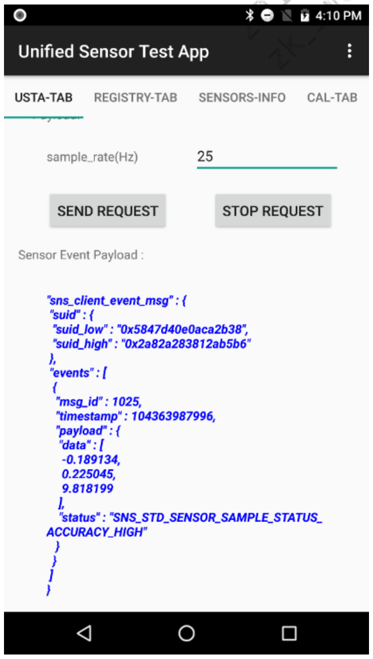
Flush Only: ☐

Max Batch: ☐

Payload:

sample_rate(Hz) 25

A sample_rate request for 25 Hz is sent to the driver. Events from the accelerometer sensor streaming on the USTA UI appear.



sample_rate(Hz) 25

Sensor Event Payload :

```
"sns_client_event_msg" : {
 "suid" : {
  "suid_low" : "0x5847d40e0aca2b38",
  "suid_high" : "0x2a82a283812ab5b6"
 },
 "events" : [
  {
   "msg_id" : 1025,
   "timestamp" : 104363987996,
   "payload" : {
    "data" : [
     -0.189134,
     0.225045,
     9.818199
    ],
    "status" : "SNS_STD_SENSOR_SAMPLE_STATUS_
ACCURACY_HIGH"
   }
  }
 ]
}
```

To view the request and events in adb logcat, issue an `adb logcat -v time` command.

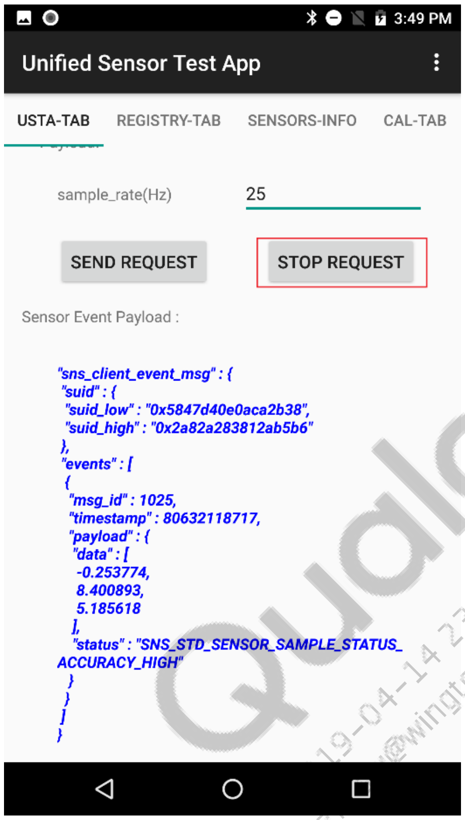
Request

```
D USTA_Native_Debug:  encoded input message for sns_std_sensor_config
is generated
D USTA_Native_Debug: payload is not empty
D USTA_Native_Debug:  encoded input message for sns_client_request_msg
is generated
D USTA_Native_Debug:
D USTA_Native_Debug: "sns_client_request_msg" : {
D USTA_Native_Debug:  "suid" : {
D USTA_Native_Debug:   "suid_low" : "0x5847d40e0aca2b38",
D USTA_Native_Debug:   "suid_high" : "0x2a82a283812ab5b6"
D USTA_Native_Debug:  },
D USTA_Native_Debug:  "msg_id" : 513,
D USTA_Native_Debug:  "susp_config" : {
D USTA_Native_Debug:   "client_proc_type" :
"SNS_STD_CLIENT_PROCESSOR_APSS",
D USTA_Native_Debug:   "delivery_type" : "SNS_CLIENT_DELIVERY_WAKEUP"
D USTA_Native_Debug:  },
D USTA_Native_Debug:  "request" : {
D USTA_Native_Debug:   "payload" : {
D USTA_Native_Debug:    "sample_rate" : 25.000000
D USTA_Native_Debug:   }
D USTA_Native_Debug:  }
D USTA_Native_Debug: }
```

Sensor events

```
D USTA_Native_Debug: accel-STMicro event message count (image (05_24))
is 1
D USTA_Native_Debug: log_message_size 344
D USTA_Native_Debug:
D USTA_Native_Debug: "sns_client_event_msg" : {
D USTA_Native_Debug:  "suid" : {
D USTA_Native_Debug:   "suid_low" : "0x5847d40e0aca2b38",
D USTA_Native_Debug:   "suid_high" : "0x2a82a283812ab5b6"
D USTA_Native_Debug:  },
D USTA_Native_Debug:  "events" : [
D USTA_Native_Debug:   {
D USTA_Native_Debug:    "msg_id" : 1025,
D USTA_Native_Debug:    "timestamp" : 107242820018,
D USTA_Native_Debug:    "payload" : {
D USTA_Native_Debug:     "data" : [
D USTA_Native_Debug:      -0.256169,
D USTA_Native_Debug:      0.244198,
D USTA_Native_Debug:      9.813411
D USTA_Native_Debug:     ],
D USTA_Native_Debug:     "status" :
"SNS_STD_SENSOR_SAMPLE_STATUS_ACCURACY_HIGH"
D USTA_Native_Debug:    }
D USTA_Native_Debug:    }
D USTA_Native_Debug:  ]
D USTA_Native_Debug: }
```

4. To stop streaming, click **STOP REQUEST**.



```
"sns_client_event_msg" : {
 "suid" : {
  "suid_low" : "0x5847d40e0aca2b38",
  "suid_high" : "0x2a82a283812ab5b6"
 },
 "events" : [
  {
   "msg_id" : 1025,
   "timestamp" : 80632118717,
   "payload" : {
    "data" : [
     -0.253774,
     8.400893,
     5.185618
    ],
    "status" : "SNS_STD_SENSOR_SAMPLE_STATUS_
ACCURACY_HIGH"
   }
  }
 ]
}
```

To view the events in adb logcat, issue a `stopRequest` command.

```
I USTA_Native_Info:  stopRequest Start
I USTA_Native_Info: stopRequest UI Mode it is
```

Related Information

“Suspend configuration fields” on page 12
“Batch specification fields” on page 14

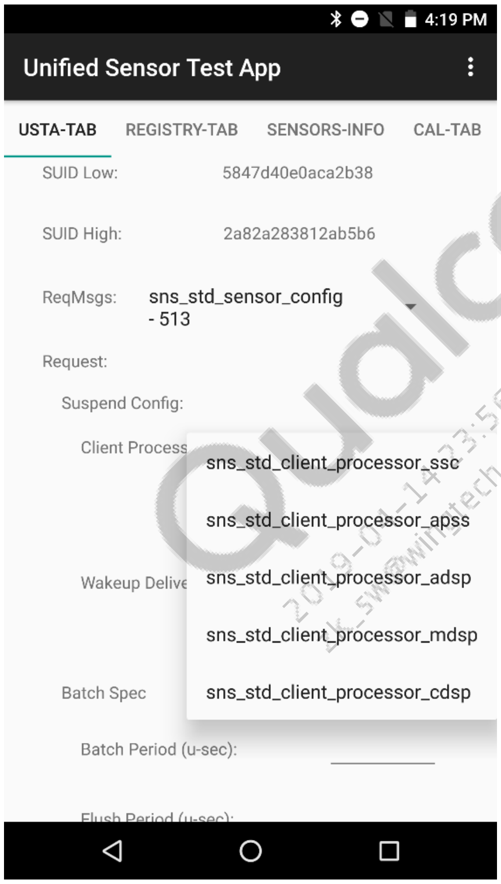
3.1 Suspend configuration fields

The Client Processor type and Wakeup Delivery type are mandatory fields and must be selected.

3.1.1 Client Processor type

The Client Processor type is the processor on which the client resides.

1. In Client Processor, click the down arrow to view the list of supported clients.



4:19 PM

Unified Sensor Test App

USTA-TAB REGISTRY-TAB SENSORS-INFO CAL-TAB

SUID Low: 5847d40e0aca2b38

SUID High: 2a82a283812ab5b6

ReqMsgs: sns_std_sensor_config - 513

Request:

Suspend Config:

Client Process

Wakeup Delive

Batch Spec

Batch Period (u-sec):

Flush Period (u-sec):

sns_std_client_processor_ssc

sns_std_client_processor_apss

sns_std_client_processor_adsp

sns_std_client_processor_mdsp

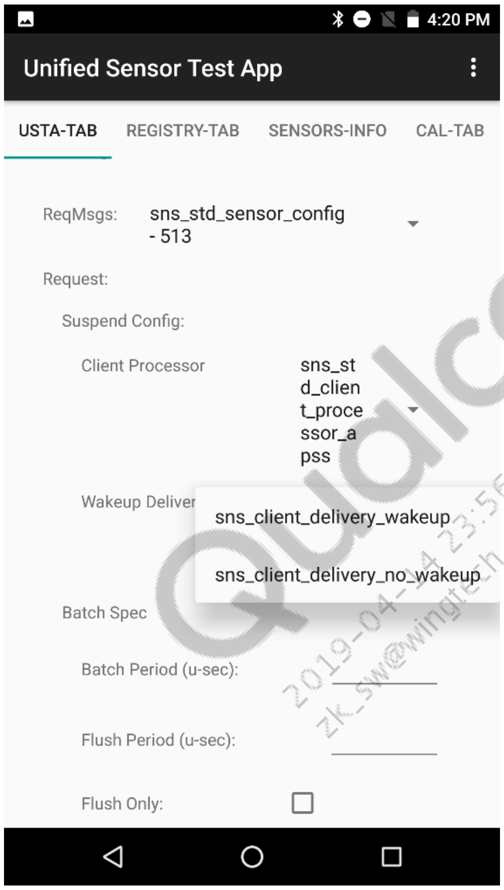
sns_std_client_processor_cdsp

2. Generally, the test application resides on an application processor; therefore, select **sns_std_client_processor_apss** as the client processor type.

3.1.2 Wakeup Delivery type

The SEE framework uses the Wakeup Delivery type to determine whether to send events to the client processor in Suspend mode.

1. In Wakeup Delivery, click the down arrow to view the list of wakeup and non-wakeup delivery types.



Unified Sensor Test App

ReqMsgs: sns_std_sensor_config - 513

Request:

Suspend Config:

Client Processor: sns_std_client_processor_apss

Wakeup Deliver

sns_client_delivery_wakeup

sns_client_delivery_no_wakeup

Batch Spec

Batch Period (u-sec):

Flush Period (u-sec):

Flush Only: ☐

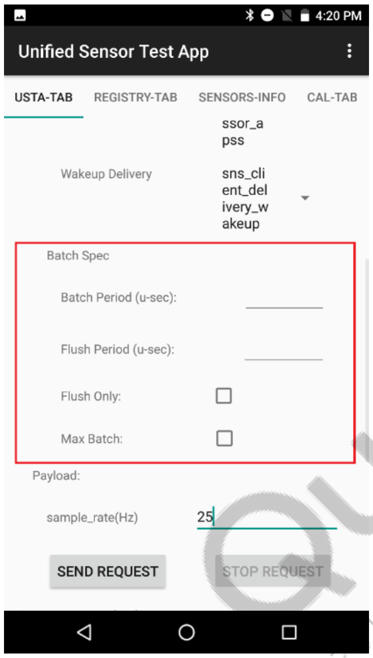
2. Select the appropriate Wakeup Delivery type.
   - sns_client_delivery_wakeup – Used if the events should be sent to the client whenever events are available. Generally, events are available at the sampling rate.
     - If a batch_period larger than the system capacity is requested, all data is sent upon capacity exhaustion.
     - The flush_period is effectively ignored, as unsent batched data does not accrue in the buffer.
   - sns_client_delivery_no_wakeup – Used if events should be sent to the client only if the client processor is awake; otherwise, batching occurs.

   When the client processor exits Suspend mode, all the events are flushed out.

3.2 Batch specification fields

The Batch Period, Flush Period, Flush Only, and Max Batch fields are part of the batch_spec message. The batch_spec message is an optional field. Therefore, it is acceptable to leave these fields blank. If the client is interested in batched data, these fields are required.

If the Flush Period, Flush Only, and Max Batch fields are provided in the batch_spec message, the Batch Period is a required field. Otherwise, the sensor does not stream.



ssor_a pss

Wakeup Delivery: sns_client_delivery_wakeup

Batch Spec

Batch Period (u-sec):

Flush Period (u-sec):

Flush Only: ☐

Max Batch: ☐

Payload:

sample_rate(Hz) 25

Batch Period

The Batch Period is an unsigned integer value that is used to set the batching interval. Events are delivered to the client according to this batching interval. Batch Period 0 indicates no batching occurs.

Flush Period

The default Flush Period is set to the Batch Period value. This parameter tells the sensor to always provide the latest flush period worth of data. If the Flush Period is less than the Batch Period, the sensor may not publish any events to clients, depending on other concurrent clients.

Flush Only

If the Flush Only field is set to True, the sensor sends data to the client only on receiving a flush request or if the sensor cannot accumulate a flush period worth of data. The default value for this field is False.

Max Batch

If the Max Batch field is set to True, the sensor operates at maximum batching capacity for all requests. The absence of this field is equivalent to `max_batch` = false. If a request has `max_batch` = true and `flush_only` = true, `flush_only` takes precedence.

4 Using USTA command line to stream sensors

USTA also provides a command-line interface. The following list of high-level command-line instructions are currently available.

- To start USTA, type the following command. No other arguments are required.

  ```
  adb shell am startservice com.qualcomm.qti.usta/
  com.qualcomm.qti.usta.core.USTACmdLineService
  ```

- To list all the sensors available on USTA, type the following command. No other arguments are required.

  ```
  adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.GETSENSORLIST
  ```

- To get attribute information for a specific sensor, type the following command.

  ```
  adb shell am broadcast -a
  com.qualcomm.qti.usta.core.intent.GETSENSORATTRIBUTES
  ```

  Select a sensor by providing the sensor handle or SUID information (both obtained from `GETSENSORLIST`). Either argument is required, that is, sensor handle (integer value with `sensorhandle` as the string name) or SUID (two string arguments with `suid_low` and `suid_high` as the string name).

  Generalized instructions:

  ```
  --ei 'sensorhandle' <sensor_handle_number>
  --es 'suid_low' <suid_low_number> --es 'suid_high' <suid_high_number>
  ```

- To close USTA, type the following command:

  ```
  adb shell am force-stop com.qualcomm.qti.usta
  ```

RELATED INFORMATION

“Handle number” on page 16
“SUID number” on page 18

4.1 Handle number

Identify the handle number for a sensor by running a `GETSENSORLIST` command. Use the handle number to stream the sensor.

The following sequence provides a detailed example of how to stream accelerometer from a USTA command line.

1. Open two command windows. In one command window, type `adb logcat -c && adb logcat -v time | tee logcat.txt` and in the other window, type the following commands in sequence.

   NOTE The order of the command-line instructions is important.

2. To start a USTA command-line service, type the following:

   ```
   adb shell am startservice com.qualcomm.qti.usta/
   com.qualcomm.qti.usta.core.USTACmdLineService
   ```

   Expected output

   ```
   Starting service: Intent { act=android.intent.action.MAIN
   cat=[android.intent.category.LAUNCHER]
   cmp=com.qualcomm.qti.usta/.core.USTACmdLineService }
   ```

3. To get the sensors list, type the following:

   ```
   adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.GETSENSORLIST
   ```

   Expected output

   ```
   Broadcasting: Intent { act=com.qualcomm.qti.usta.core.intent.GETSENSORLIST
   flg=0x400000 }
   Broadcast completed: result=0
   ```

   In an adb logcat window, all the sensor types with handle numbers appear.

   ```
   ...
   D/USTA_APP_DEBUG( 9499): Sensor Handle:21 Sensor Name:motion_detect-
   STMicro Sensor SUID Low:304c23e388e96faf Sensor SUID High:320a2b36a64d3fa7
   D/USTA_APP_DEBUG( 9499): Sensor Handle:22 Sensor Name:gyro-STMicro Sensor
   SUID Low:be4bfe881113e717 Sensor SUID High:11ea006282d7f9c
   D/USTA_APP_DEBUG( 9499): Sensor Handle:23 Sensor Name:accel-STMicro Sensor
   SUID Low:5847d40e0aca2b38 Sensor SUID High:2a82a283812ab5b6
   ...
   ```

4. To get the sensor attributes for the selected sensor handle, type the following:

   ```
   adb shell am broadcast -a
   com.qualcomm.qti.usta.core.intent.GETSENSORATTRIBUTES --ei 'sensorhandle'
   23
   ```

   Expected output

   ```
   Broadcasting: Intent
   { act=com.qualcomm.qti.usta.core.intent.GETSENSORATTRIBUTES flg=0x400000
   (has extras) }
   Broadcast completed: result=0
   ```

In an adb logcat window, all the attributes for the selected sensor appear.

5. To get the request message for the selected sensor handle, type the following:

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.GETREQUESTMSGS
--ei 'sensorhandle' 23
```

Expected output

```
Broadcasting: Intent
{ act=com.qualcomm.qti.usta.core.intent.GETREQUESTMSGS flg=0x400000 (has
extras) }
Broadcast completed: result=0
```

In an adb logcat window, all the supported request parameters for the selected sensor appear.

6. To enable the selected sensor, type the following:

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.ENABLE --ei
'sensorhandle' 23 --ei 'msg_id' 513 --ei 'client' 0 --ei 'wakeup' 0 --ei
'batch_period' 0  --ei 'flush_period' 0 --ef 'sample_rate' 2.0 --es
'logfile' 'sensorLog.json'
```

Expected output

```
Broadcasting: Intent { act=com.qualcomm.qti.usta.core.intent.ENABLE
flg=0x400000 (has extras) }
Broadcast completed: result=0
```

The key parameters are `sensorhandle 23`, `msg_id 513` (for `sns_std_sensor_config`), similar to the USTA UI options.

In an adb logcat window, the accelerometer events streaming appears.

```
D/USTA_Native_Debug( 9499): log_message_size 344
D/USTA_Native_Debug( 9499):
D/USTA_Native_Debug( 9499): "sns_client_event_msg" : {
D/USTA_Native_Debug( 9499):  "suid" : {
D/USTA_Native_Debug( 9499):   "suid_low" : "0x5847d40e0aca2b38",
D/USTA_Native_Debug( 9499):   "suid_high" : "0x2a82a283812ab5b6"
D/USTA_Native_Debug( 9499):  },
D/USTA_Native_Debug( 9499):  "events" : [
D/USTA_Native_Debug( 9499):   {
D/USTA_Native_Debug( 9499):    "msg_id" : 1025,
D/USTA_Native_Debug( 9499):    "timestamp" : 178025383752,
D/USTA_Native_Debug( 9499):    "payload" : {
D/USTA_Native_Debug( 9499):     "data" : [
D/USTA_Native_Debug( 9499):      -0.184346,
D/USTA_Native_Debug( 9499):      0.093370,
D/USTA_Native_Debug( 9499):      9.763135
D/USTA_Native_Debug( 9499):     ],
D/USTA_Native_Debug( 9499):     "status" :
"SNS_STD_SENSOR_SAMPLE_STATUS_ACCURACY_HIGH"
D/USTA_Native_Debug( 9499):    }
D/USTA_Native_Debug( 9499):   }
D/USTA_Native_Debug( 9499):  ]
D/USTA_Native_Debug( 9499): }
D/USTA_Native_Debug( 9499): accel-STMicro event message count (image
(05_24)) is 1
```

7. To disable the selected sensor, type the following:

   ```
   adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.DISABLE --ei
   'sensorhandle' 23
   ```

   Expected output

   ```
   Broadcasting: Intent { act=com.qualcomm.qti.usta.core.intent.DISABLE
   flg=0x400000 (has extras) }
   Broadcast completed: result=0
   ```

   In an adb logcat window, logs shown indicate a disabling request for `sensorhandle 23`.

   ```
   I/USTA_APP_INFO( 9499): DISABLE intent
   D/USTA_APP_DEBUG( 9499): stop request from command line for sensor 23
   I/USTA_Native_Info( 9499):  stopRequest Start
   I/USTA_Native_Info( 9499): stopRequest COMMAND_LINE Mode it is
   I/USTA_Native_Info( 9499):  stopRequest End
   D/USTA_APP_DEBUG( 9499): after stop request for the sensorhandle 23
   ```

8. To stop USTA command-line service, type the following:

   ```
   adb shell am stopservice com.qualcomm.qti.usta/
   com.qualcomm.qti.usta.core.USTACmdLineService
   ```

   Expected output

   ```
   Stopping service: Intent { act=android.intent.action.MAIN
   cat=[android.intent.category.LAUNCHER]
   cmp=com.qualcomm.qti.usta/.core.USTACmdLineService }
   Service stopped
   ```

4.2 SUID number

Identify the SUID for a sensor by running a `GETSENSORLIST` command. Use the SUID to stream the sensor.

```
adb shell am startservice com.qualcomm.qti.usta/
com.qualcomm.qti.usta.core.USTACmdLineService
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.GETSENSORLIST
```

In an adb logcat window, all sensor types listed with SUIDs for accelerator appear along with the handle number.

```
...
D/USTA_APP_DEBUG( 9499): Sensor Handle:21 Sensor Name:motion_detect-STMicro
Sensor SUID Low:304c23e388e96faf Sensor SUID High:320a2b36a64d3fa7
D/USTA_APP_DEBUG( 9499): Sensor Handle:22 Sensor Name:gyro-STMicro Sensor
SUID Low:be4bfe881113e717 Sensor SUID High:11ea006282d7f9c
D/USTA_APP_DEBUG( 9499): Sensor Handle:23 Sensor Name:accel-STMicro Sensor
SUID Low:5847d40e0aca2b38 Sensor SUID High:2a82a283812ab5b6
...
```

The following example shows a request made with a SUID (low and high) for accelerometer.

```
adb shell am broadcast -a
com.qualcomm.qti.usta.core.intent.GETSENSORATTRIBUTES --es 'suid_low'
5847d40e0aca2b38 --es 'suid_high' 2a82a283812ab5b6
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.GETREQUESTMSGS --
es 'suid_low' 5847d40e0aca2b38 --es 'suid_high' 2a82a283812ab5b6
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.ENABLE --es
'suid_low' 5847d40e0aca2b38 --es 'suid_high' 2a82a283812ab5b6 --ei 'msg_id'
513 --ei 'client' 0 --ei 'wakeup' 0 --ei 'batch_period' 0 --ei 'flush_period'
0 --ef 'sample_rate' 26.0 --es 'logfile' 'sensorLog.json'
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.DISABLE --es
'suid_low' 5847d40e0aca2b38 --es 'suid_high' 2a82a283812ab5b6
adb shell am stopservice com.qualcomm.qti.usta/
com.qualcomm.qti.usta.core.USTACmdLineService
```

5 Other options and menu settings

USTA supports Disable Logging, Streaming Status, and Force Close App.

5.1 Disable Logging

Disable Logging controls USTA logging, such as adb logcat logs and logging into /mnt/sdcard/.

From the UI

To disable logs from the UI, select **Disable Logging** before clicking the USTA tab.

Change the logging status at any time during the execution of USTA by selecting or clearing **Disable Logging**.

From the command line

- To disable logging from a command line, type the following command before running the start service command:

  ```
  adb shell setprop sensors.log.disable 1
  ```

- To disable logging during the execution of USTA from a command line, type the following command-line instruction. Switch between enable and disable logging at any time.
  - To disable logging, type the following:

    ```
    adb shell am broadcast -a
    com.qualcomm.qti.usta.core.intent.DISABLELOGGING --ei 'disable_logging'
    1
    ```

  - To enable logging, type the following:

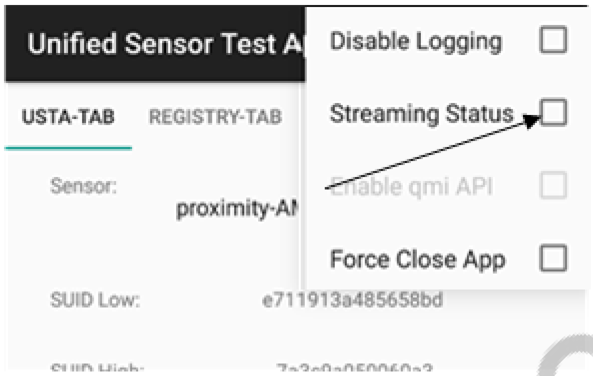
    ```
    adb shell am broadcast -a
    com.qualcomm.qti.usta.core.intent.DISABLELOGGING --ei 'disable_logging'
    0
    ```

In both the cases, logging is enabled by default.

Using a command line is recommended when USTA is used for power measurement purposes.

5.2 Streaming Status

Streaming Status controls whether sensor event messages appear on the UI. By default, events appear on USTA. To disable event messages on the USTA UI, clear the **Streaming Status** checkbox.



5.3 Force Close App

Force Close App closes USTA and removes the application from the recent applications window.

6 Registry functionality

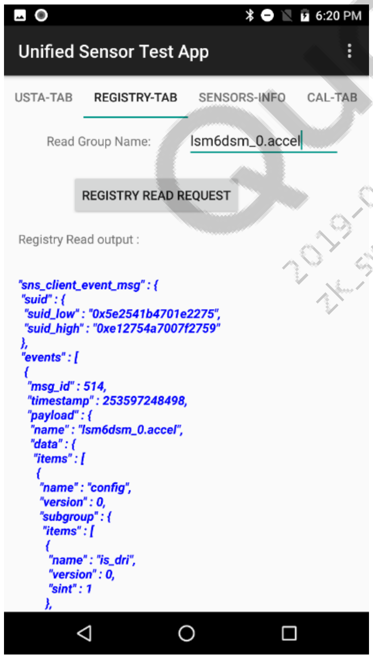
USTA supports reading and writing/modifying the registry based on the registry group name.

From the UI

- Registry Read

  Type a group name in the **Read Group Name** field in which you want to read, and click **Registry Read Request**. Registry details appear on the USTA UI screen.

  For example, for accelerometer (LSM6DSM), **lsm6dsm_0.accel** in the group name field shows the event message on the UI and in adb logcat.



```
"sns_client_event_msg" : {
 "suid" : {
  "suid_low" : "0x5e2541b4701e2275",
  "suid_high" : "0xe12754a7007f2759"
 },
 "events" : [
  {
   "msg_id" : 514,
   "timestamp" : 253597248498,
   "payload" : {
    "name" : "lsm6dsm_0.accel",
    "data" : {
     "items" : [
      {
       "name" : "config",
       "version" : 0,
       "subgroup" : {
        "items" : [
         {
          "name" : "is_dri",
          "version" : 0,
          "sint" : 1
         },
```

- Registry Write

  Type a group name in **Write Group Name** in which you want to write an item. Type the data item name in **Write Data Name** and version number in **Version**. (The version number must be one greater than the previous one.). After all the fields are complete, click **Registry Write request**.

  To verify whether the registry write for this data item is successful, send a **Registry Read** for the group name and cross-check the values entered.

From the command line

Registry groups can be read from a command line. Similar commands must be used for a command line, with the exception that the `sensor_handle` is used for a registry sensor.

Generic commands

- Read

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.ENABLE --ei
'sensorhandle' <registry_sensor_Handle_number> --ei 'msg_id' 512 --ei
'client' 1 --ei 'wakeup' 0 --es 'name' <registry_group_name>
```

- Write

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.ENABLE --ei
'sensorhandle' <registry_sensor_Handle_number> --ei 'msg_id' 513 --ei
'client' 1 --ei 'wakeup' 0 --es 'sub_message_name' <group_name>  --es
'data_name' <data_item_name> --ei 'version' <version_number> --es
'oneof_name' <one_of_filed_string> --es 'one_of_value' <one_of_value>
```

For example, the following command is used for lsm6dsm_0.accel.config:

- Read

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.ENABLE --ei
'sensorhandle' 15 --ei 'msg_id' 512 --ei 'client' 1 --ei 'wakeup' 0 --es
'name' lsm6dsm_0.accel.config  // Registry Read
```

a. To start the USTA command-line service, type the following:

```
adb shell am startservice com.qualcomm.qti.usta/
com.qualcomm.qti.usta.core.USTACmdLineService
```

b. To get the sensor list and find the registry sensor, type the following:

```
adb shell am broadcast -a
com.qualcomm.qti.usta.core.intent.GETSENSORLIST
…
      D/USTA_APP_DEBUG( 9499): Sensor Handle:15 Sensor Name:registry-
qualcomm Sensor SUID Low:5e2541b4701e2275 Sensor SUID
High:e12754a7007f2759
       …
```

c. To read a registry group using a registry sensor, for example, lsm6dsm_0.accel.config, type the following:

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.ENABLE --ei
'sensorhandle' 15 --ei 'msg_id' 512 --ei 'client' 1 --ei 'wakeup' 0 --
es 'name' lsm6dsm_0.accel.config
```

An adb logcat window shows the registry output.

```
D/USTA_Native_Debug( 9499): "sns_client_event_msg" : {
D/USTA_Native_Debug( 9499):  "suid" : {
D/USTA_Native_Debug( 9499):   "suid_low" : "0x5e2541b4701e2275",
D/USTA_Native_Debug( 9499):   "suid_high" : "0xe12754a7007f2759"
D/USTA_Native_Debug( 9499):  },
D/USTA_Native_Debug( 9499):  "events" : [
D/USTA_Native_Debug( 9499):   {
D/USTA_Native_Debug( 9499):    "msg_id" : 514,
D/USTA_Native_Debug( 9499):    "timestamp" : 304728899368,
D/USTA_Native_Debug( 9499):    "payload" : {
D/USTA_Native_Debug( 9499):     "name" : "lsm6dsm_0.accel.config",
D/USTA_Native_Debug( 9499):     "data" : {
D/USTA_Native_Debug( 9499):      "items" : [
D/USTA_Native_Debug( 9499):       {
D/USTA_Native_Debug( 9499):        "name" : "is_dri",
D/USTA_Native_Debug( 9499):        "version" : 0,
D/USTA_Native_Debug( 9499):        "sint" : 1
```

d. To disable the registry sensor, type the following:

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.DISABLE --
ei 'sensorhandle' 15
```

e. To stop the USTA command-line service, type the following:

```
adb shell am stopservice com.qualcomm.qti.usta/
com.qualcomm.qti.usta.core.USTACmdLineService
```

- Write

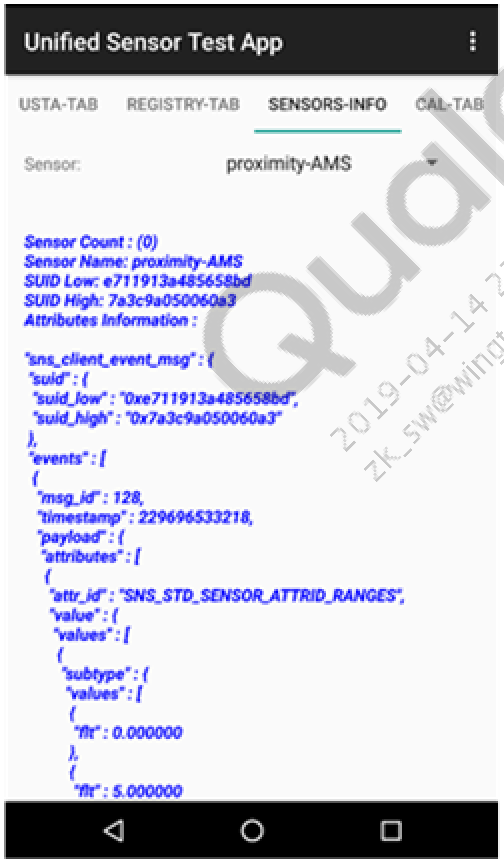
  To write to a registry group using a registry sensor, for example, lsm6dsm_0.accel.config, type the following:

```
adb shell am broadcast -a com.qualcomm.qti.usta.core.intent.ENABLE --ei
'sensorhandle' 15 --ei 'msg_id' 513 --ei 'client' 1 --ei 'wakeup' 0 --es
'sub_message_name' lsm6dsm_0.accel.config  --es 'data_name' is_dri --ei
'version' 4 --es 'oneof_name' sint --es 'one_of_value' 20
```

7 Sensor attributes

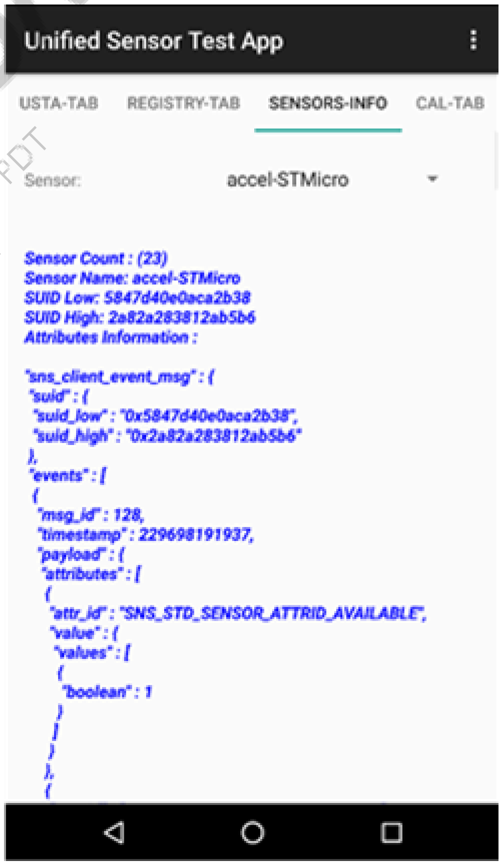
The SENSORS-INFO tab shows the attributes for each sensor identified by the USTA UI. By default, a sensor with a zero (0) handle displays its attributes. To view the attributes of a specific sensor, select a sensor from the drop-down list.

The following examples show attributes for a proximity sensor and accelerometer sensor.



```
Sensor Count : (0)
Sensor Name: proximity-AMS
SUID Low: e711913a485658bd
SUID High: 7a3c9a050060a3
Attributes Information :

"sns_client_event_msg" : {
 "suid" : {
  "suid_low" : "0xe711913a485658bd",
  "suid_high" : "0x7a3c9a050060a3"
 },
 "events" : [
  {
   "msg_id" : 128,
   "timestamp" : 229696533218,
   "payload" : {
    "attributes" : [
     {
      "attr_id" : "SNS_STD_SENSOR_ATTRID_RANGES",
      "value" : {
       "values" : [
        {
         "subtype" : {
          "values" : [
           {
            "flt" : 0.000000
           },
           {
            "flt" : 5.000000
```



```
Sensor Count : (23)
Sensor Name: accel-STMicro
SUID Low: 5847d40e0aca2b38
SUID High: 2a82a283812ab5b6
Attributes Information :

"sns_client_event_msg" : {
 "suid" : {
  "suid_low" : "0x5847d40e0aca2b38",
  "suid_high" : "0x2a82a283812ab5b6"
 },
 "events" : [
  {
   "msg_id" : 128,
   "timestamp" : 229698191937,
   "payload" : {
    "attributes" : [
     {
      "attr_id" : "SNS_STD_SENSOR_ATTRID_AVAILABLE",
      "value" : {
       "values" : [
        {
         "boolean" : 1
        }
       ]
      }
     },
     {
```

A References

A.1 Acronyms and terms

Acronym or term	Definition
SEE	Sensors Execution Environment
SUID	Sensor unique identifier
USTA	Unified Sensor Test Application