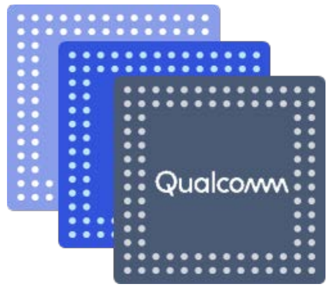
Qualcomm Technologies, Inc.

Adding a Custom Algorithm with Sensors Execution Environment (SEE)

80-P9301-67 Rev. B



Confidential and Proprietary – Qualcomm Technologies, Inc.

Revision History

Revision	Date	Description
A	June 2017	Initial release
B	March 2019	Numerous updates have been made to this document; it should be read in its entirety

Contents

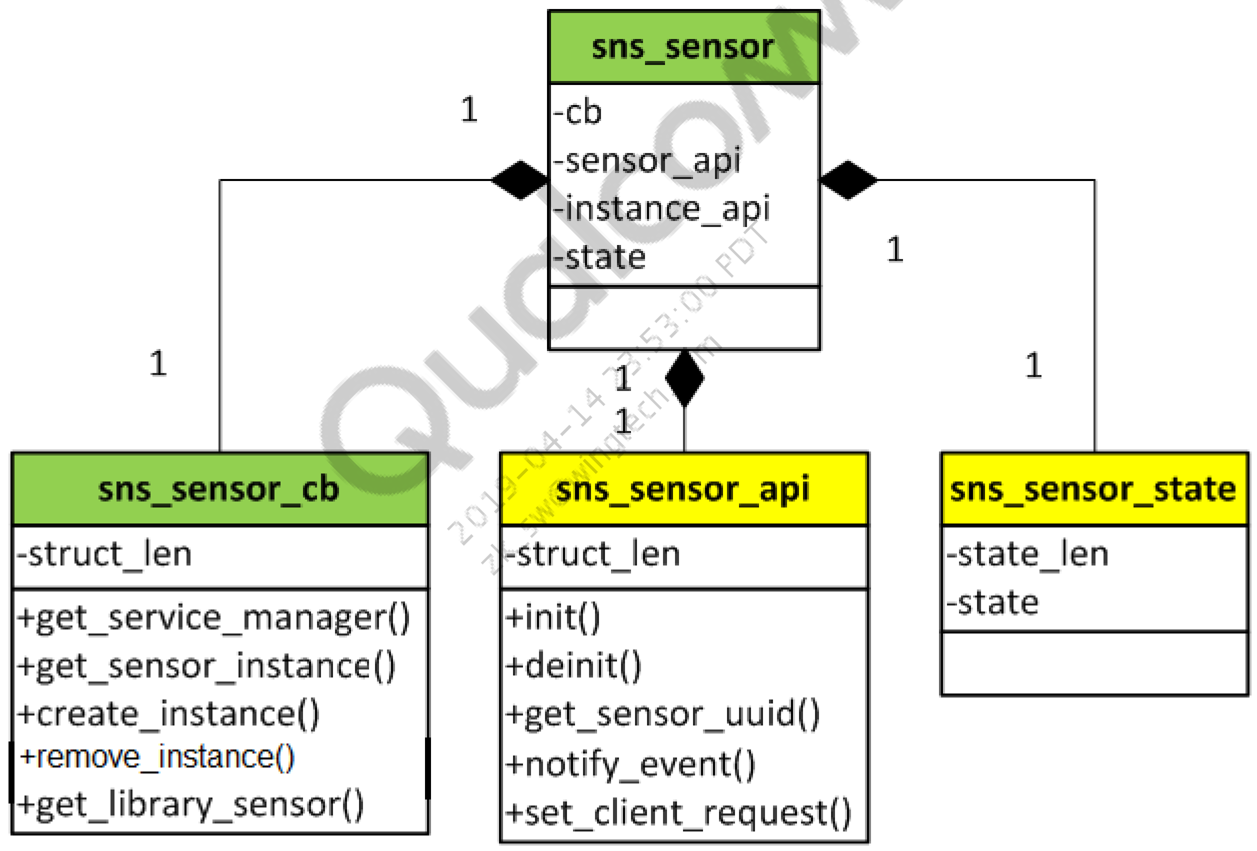
Introduction

Objectives

- Describes how customers add a custom algorithm with Sensors Execution Environment (SEE)
- Discusses OEM1, a template algorithm that is provided by Qualcomm Technologies, Inc. (QTI) as reference, and how to test it
- Assumes basic understanding of SEE framework and some knowledge of sensor and sensor instance APIs
  - APIs are defined in the following files:
    - \<root>\ssc\inc\sns_sensor.h
    - \<root>\ssc\inc\sns_sensor_instance.h
    - \<root>\ssc\inc\sns_register.h

Summary of Sensor and Sensor Instance APIs

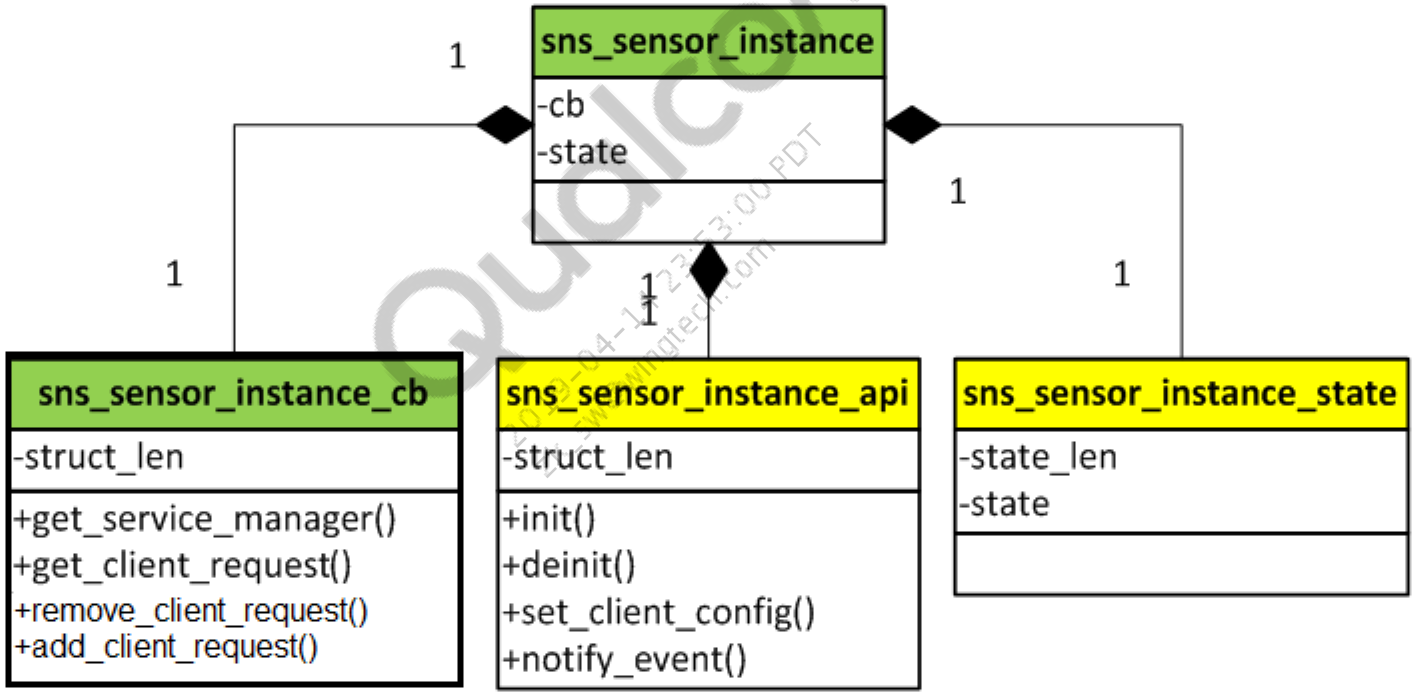
Sensor API


sns_sensor
-cb
-sensor_api
-instance_api
-state
1
1
1
1
1
1
sns_sensor_cb
-struct_len
+get_service_manager()
+get_sensor_instance()
+create_instance()
+remove_instance()
+get_library_sensor()
sns_sensor_api
-struct_len
+init()
+deinit()
+get_sensor_uuid()
+notify_event()
+set_client_request()
sns_sensor_state
-state_len
-state

SEE Sensor API – sns_sensor.h

- `sns_sensor` struct defines:
  - `sns_sensor_cb` – Callbacks in the SEE framework available to a sensor
    - Gets the service manager handle (`.get_service_manager()`)
    - Gets the next instance for the sensor (`.get_sensor_instance()`)
    - Creates a new instance (`.create_instance()`)
    - Removes an existing instance (`.remove_instance()`)
    - Gets the next sensor supported in this sensor's library (`.get_library_sensor()`)
  - `sns_sensor_api` – Functions that every sensor must implement and are called only by the SEE framework
    - Initializes a sensor (`.init()`)
      - Called during the sensor registration
    - Destroys a sensor (`.deinit()`)
      - Called when the sensor reports any failure error code
    - Gets the unique sensor identifier (`.get_sensor_uid()`)
    - Notifies a sensor that an event is available from a dependent sensor (`.notify_event()`)
    - Updates a client request for a sensor (`.set_client_request()`)
  - `sns_sensor_state` – Private state maintained by a sensor
    - Memory for the sensor state is allocated by the SEE framework during the sensor registration

Sensor Instance API



SEE Sensor API – sns_sensor_instance.h

- `sns_sensor_instance` struct defines:
  - `sns_sensor_instance_cb` – Callbacks in the SEE framework available to a sensor instance
    - Gets the service manager handle (`.get_service_manager()`)
    - Gets the next client request associated with an instance (`.get_client_request()`)
    - Removes a client request handled by an instance (`.remove_client_request()`)
    - Adds a client request handled by an instance (`.add_client_request`)
  - `sns_sensor_instance_state` – Private state maintained by a sensor instance
    - SEE framework allocates the memory for a sensor instance state during the creation of a sensor instance
- `sns_sensor_instance_api` – Functions that every sensor instance must implement and are called by its sensor or the SEE framework
  - Initializes a sensor instance (`.init()`)
    - Called by the SEE framework when an instance is created
  - Destroys a sensor instance (`.deinit()`)
    - Called by the SEE framework when its sensor requests a `remove_instance()`.
  - Sets instance configuration (`.set_client_config()`)
    - Called by the sensor to set the instance configuration
  - Notifies an instance that an event is available from a dependent sensor (`.notify_event()`)
    - Called by the SEE framework

SEE Sensor API – sns_register.h

- Each driver library implements a registration function of type `sns_register_sensors()`.
  - `sns_register_cb struct` defines:
    - Callback to initialize a sensor (`.init_sensor()`)
- This function calls `init_sensor()` on each supported sensor within the library.

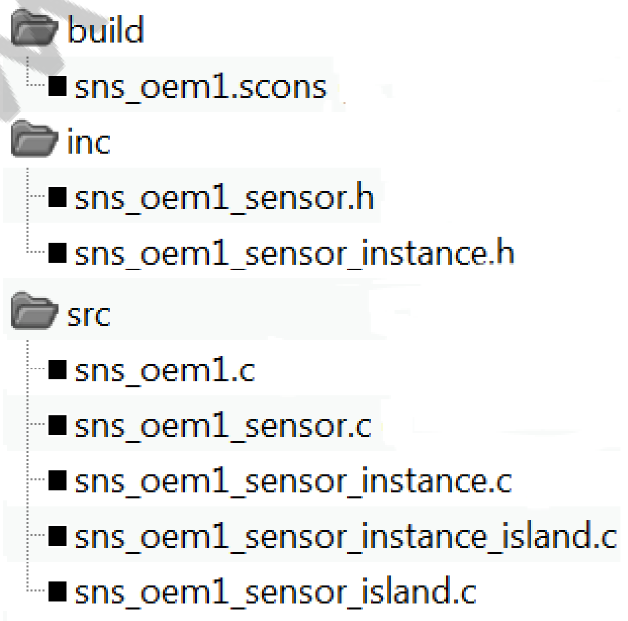
Algorithm Development

Sensor and Sensor Instance APIs

Files	Description
sns_<algo_name>.c	Function to register algorithm library
sns_<algo_name>_sensor.c sns_<algo_name>_sensor.h	▪ Normal mode APIs for algorithm ▪ Data types for algorithm
sns_<algo_name>_sensor_island.c	Island mode functions for algorithm
sns_<algo_name>_sensor_instance.c sns_<algo_name>_sensor_instance.h	▪ Normal mode functions for the sensor instance ▪ Sensor instance data types for the algorithm
sns_<algo_name>_sensor_instance_island.c	Island mode functions for the algorithm instance

Writing a Custom Algorithm (OEM1 Template)

- Algorithms (like other sensors) reside under the slpi_proc\ssc\sensors\<algo_name> directory with SEE.
- OEM1 is provided as a sample algorithm in the default QTI SLPI code as a template algorithm.
  - slpi_proc\ssc\sensors\oem1\
  - Proto file for the OEM1 algorithm is placed in slpi_proc\ssc\sensors\pb\sns_oem1.proto.



```
build
  sns_oem1.scons
inc
  sns_oem1_sensor.h
  sns_oem1_sensor_instance.h
src
  sns_oem1.c
  sns_oem1_sensor.c
  sns_oem1_sensor_instance.c
  sns_oem1_sensor_instance_island.c
  sns_oem1_sensor_island.c
```

Defining a .proto File for Custom Algorithm

- The .proto file defines the messages used by the custom algorithm, using the protocol buffer specification language.
- Custom algorithms can take advantage of standard messages defined in slpi_proc\ssc\sensors\pb\sns_std_sensor.proto.
- Message IDs are defined in sns_oem1.proto.

```
enum sns_oem1_msgid
{
  option (nanopb_enumopt).long_names = false;
  SNS_OEM1_MSGID_SNS_STD_SENSOR_CONFIG = 512;
  SNS_OEM1_MSGID_SNS_OEM1_DATA = 1024;
}
```

Defining a .proto File for Custom Algorithm (cont.)

- OEM1 sensor requires only the sampling rate

```
// Sensor stream configuration request
// or configuration change message
message sns_std_sensor_config
{
  // Sample rate in Hz.
  required float sample_rate = 1;
}
```

- Output data defined in sns_oem1.proto generated by the OEM1 algorithm

```
message sns_oem1_data
{
  // oem1 Vector along axis x,y,z in m/s2
  repeated float oem1 = 1 [(nanopb).max_count = 3];
  // Accuracy of the data
  required sns_std_sensor_sample_status accuracy = 2;
}
```

OEM1 Sensor APIs

- sns_oem1_sensor.c
  - sns_oem1_init
    - Publish most of the attributes in this API
  - sns_oem1_deinit
- sns_oem1_sensor_island.c
  - sns_oem1_get_sensor_uid
  - sns_oem1_set_client_request
  - sns_oem1_notify_event
    - Publish available in SNS_STD_SENSOR_ATTRID_AVAILABLE

OEM1 Sensor Instance APIs

- sns_oem1_sensor_instance.c
  - sns_oem1_inst_init
  - sns_oem1_inst_deinit
- sns_oem1_sensor_instance_island.c
  - sns_oem1_inst_set_client_config
  - sns_oem1_inst_notify_event

Key Functions

Dependent sensors

- sns_oem1_init in sns_oem1_sensor.c

```
sns_oem1_init(sns_sensor *const this)
{
  . . .
  . . .
  state->suid_search[state->search_count].data_type_str = "accel";
```

- OEM1 can be updated to use any other sensor instead of accelerometer by replacing the appropriate attribute SNS_STD_SENSOR_ATTRID_TYPE of the requisite sensor
  - Gyroscope

    “state->suid_search[state->search_count].data_type_str = “**gyro**";
  - Magnetometer

    “state->suid_search[state->search_count].data_type_str = “**mag**";

Key Functions (cont.)

Algorithm logic

- `sns_oem1_inst_notify_event()` in sns_oem1_sensor_instance_island.c

```
         //This is dummy logic for OEM1 demonstration purposes
       //OEMs can replace with their algo logic
           if (0 < temp[2])
           {
                       accel_payload[0]=100;
                       accel_payload[1]=temp[1];
                       accel_payload[2]=temp[2];
           }
           else
           {
                       accel_payload[0]=9;
                       accel_payload[1]=temp[1];
                       accel_payload[2]=temp[2];
           }
```

- `temp[0-2]` contains input to the algorithm from the dependent sensor, that is, x, y, and z axes of accelerometer data in the default implementation
- `accel_payload[0-2]` contains the output of the algorithm generated by OEM1
- OEMs can replace this dummy logic with actual algorithm code

Common Customizations

Common Customizations

- OEM1 is designed to highlight various common customizations that the SEE algorithm developer may need, for example:
  - Supporting output as streaming or on-change
  - Algorithm requiring sensor input at a fixed rate
  - Registry support (reading certain configuration parameters)

Custom flag (in sns_oem1_sensor.h)	Description
#define OEM1_SUPPORT_ **DIRECT_SENSOR_REQUEST**	▪ With this flag enabled: ▪ OEM1 algorithm requests the data from accel sensor directly ▪ The rate at which algorithm receives input sensor samples (accel) is dependent on ODR of accel driver, e.g., if there is a concurrent client for accel sensor at a higher rate, algorithm would receive accel data at higher rate) ▪ With this flag disabled: ▪ OEM1 requests accel data via QTI-provided resampler module which always confirms the rate is same as requested and it is oblivious to any other concurrent requests
#define OEM1_SUPPORT_ **EVENT_TYPE**	▪ By default, OEM1 output is streaming type, i.e., every time the algorithm receives input from accel, it will calculate and generate event. ▪ If requirement for the algorithm is to have on-change/event output, then this flag should be enabled; with this, the OEM1 algorithm reports the data only if output changes (and not every output).
#define OEM1_SUPPORT_ **REGISTRY**	▪ This flag can be enabled if the requirement is for the algorithm to read certain configuration parameters from registry. ▪ Besides enabling OEM1_SUPPORT_REGISTRY on the SLPI side, the sns_oem1.json file should also be placed on the AP side where the rest of the JSON files are present. adb push sns_oem1.json /vendor/etc/sensors/config

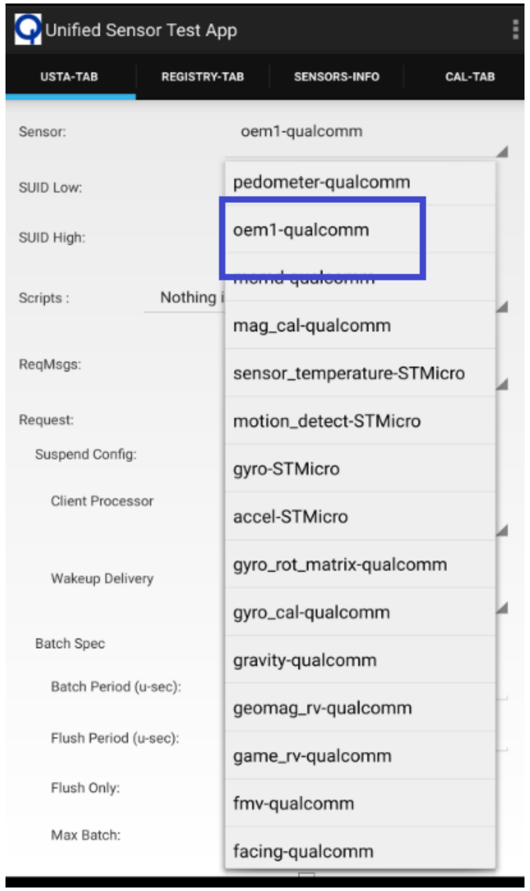
Common Customizations (cont.)

- For more details on these customizations and to understand associated changes required in the algorithm, it is recommended the SEE algorithm developer understand the OEM1 template code thoroughly and map it to the requirements of the algorithm. It is recommended to study the code around these flags:
  - Resampler vs. direct requests (OEM1_SUPPORT_DIRECT_SENSOR_REQUEST)
    - Refer to how SNS_RESAMPLER_MSGID_SNS_RESAMPLER_CONFIG is handled for resampler requests vs. SNS_STD_SENSOR_MSGID_SNS_STD_SENSOR_CONFIG for direct resampler
      - “resampler_config.resampled_rate” defined for resampler configuration
  - On-change vs. streaming (OEM1_SUPPORT_EVENT_TYPE)
    - SNS_STD_SENSOR_MSGID_SNS_STD_ON_CHANGE_CONFIG is handled for on-change vs. SNS_STD_SENSOR_MSGID_SNS_STD_SENSOR_CONFIG for streaming
- By default, OEM1 is shipped with resampler enabled and streaming output, i.e., OEM1_SUPPORT_DIRECT_SENSOR_REQUEST and OEM1_SUPPORT_EVENT_TYPE disabled.

Testing Custom Algorithm

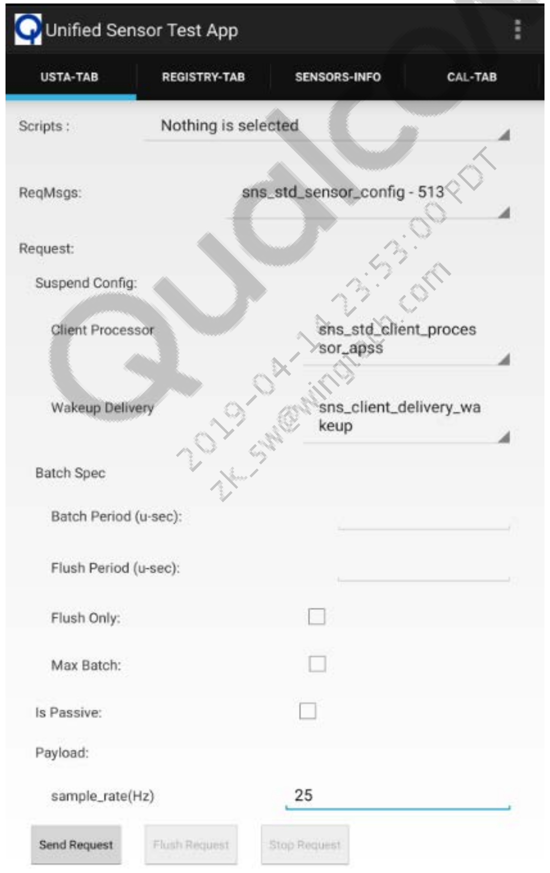
Unified Sensor Test Application (USTA)

- OEM1 can be tested from USTA as shown.

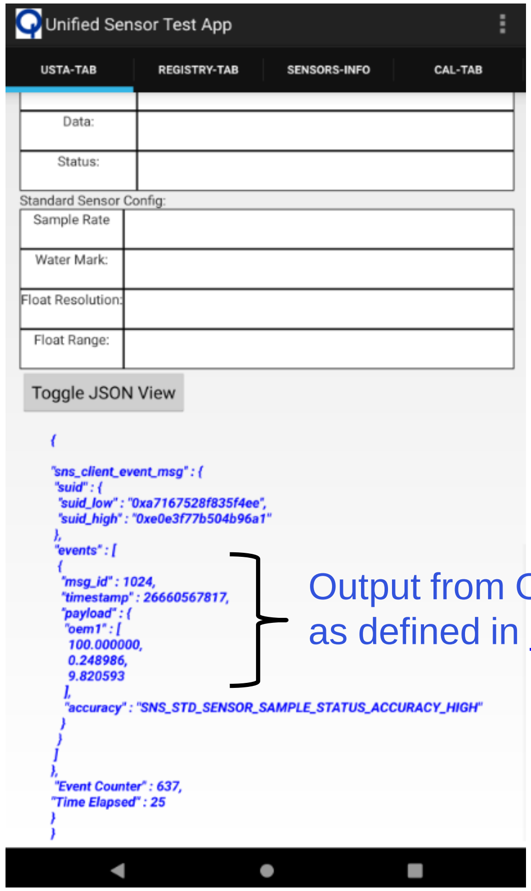


Sensor: oem1-qualcomm
SUID Low:
SUID High:
Scripts : Nothing i
ReqMsgs:
Request:
Suspend Config:
Client Processor
Wakeup Delivery
Batch Spec
Batch Period (u-sec):
Flush Period (u-sec):
Flush Only:
Max Batch:

pedometer-qualcomm
oem1-qualcomm
memd-qualcomm
mag_cal-qualcomm
sensor_temperature-STMicro
motion_detect-STMicro
gyro-STMicro
accel-STMicro
gyro_rot_matrix-qualcomm
gyro_cal-qualcomm
gravity-qualcomm
geomag_rv-qualcomm
game_rv-qualcomm
fmv-qualcomm
facing-qualcomm



Scripts : Nothing is selected
ReqMsgs: sns_std_sensor_config - 513
Request:
Suspend Config:
Client Processor sns_std_client_processor_apss
Wakeup Delivery sns_client_delivery_wakeup
Batch Spec
Batch Period (u-sec):
Flush Period (u-sec):
Flush Only:
Max Batch:
Is Passive:
Payload:
sample_rate(Hz) 25



Data:
Status:
Standard Sensor Config:
Sample Rate
Water Mark:
Float Resolution:
Float Range:

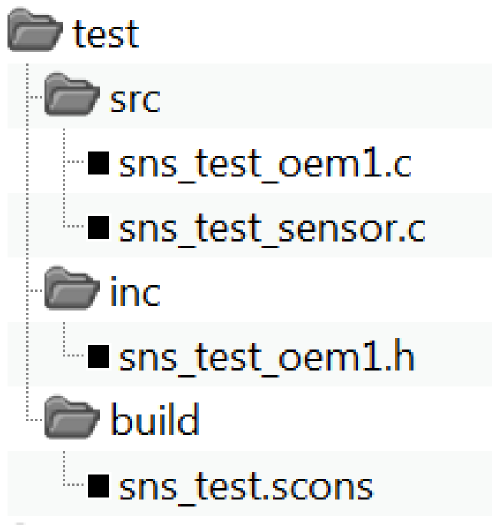
```
{

"sns_client_event_msg" : {
 "suid" : {
  "suid_low" : "0xa7167528f835f4ee",
  "suid_high" : "0xe0e3f77b504b96a1"
 },
 "events" : [
  {
   "msg_id" : 1024,
   "timestamp" : 26660567817,
   "payload" : {
    "oem1" : [
     100.000000,
     0.248986,
     9.820593
    ],
    "accuracy" : "SNS_STD_SENSOR_SAMPLE_STATUS_ACCURACY_HIGH"
   }
  }
 ]
},
 "Event Counter" : 637,
"Time Elapsed" : 25
}
}
```

$\left.\right\}$ Output from OEM1 algorithm as defined in Key Functions

OEM1 Test Sensor (Resident on SLPI)

- SLPI resident test sensor can be used for OEM1 testing; this is only recommended as an intermediate step if USTA is not available for some reason on a particular platform.
- The test sensor located in slpi_proc\ssc\sensors\test is enhanced to enable OEM1 algorithm testing from the test sensor.
- OEMs can similarly add new files for custom algorithms.
  - New files
    - sns_test_oem1.c
    - sns_test_oem1.h
  - Update
    - sns_test_sensor.c
    - sns_test.scons
- Use the following build flags in the build command:
  - `SNS_DELAY_INIT` ← To view the boot-up time logs and OEM1 initialization logs
  - `SNS_TEST_OEM1` ← To enable test sensor to stream OEM1



```
test
  src
    sns_test_oem1.c
    sns_test_sensor.c
  inc
    sns_test_oem1.h
  build
    sns_test.scons
```

Important: To reiterate, USTA is the recommended method to validate OEM1 or any custom algorithm.

Testing OEM1

- On successful integration of OEM1 in the SLPI build, logs similar to the following appear:

```
[0053/0000] | MSG | 00:31:35.924 | SNS/Low | [ sns_event_service.c    303] Event published by Instance b22afb90 (accel) for Stream
b200ea20 (to Instance of oem1)(len 17)
```
← ***Event published from accel sensor instance to data stream for OEM1***

```
[0053/0000] | MSG | 00:31:35.924 | SNS/Low | [ sns_event_service.c    450] Notifying Sensor 'oem1' of event from 'accel‘
```
← ***OEM1 notified of the event***

```
[0053/0000] | MSG | 00:31:35.924 | SNS/Low | [sns_oem1_sensor_instance_island.c     42] sns_oem1_inst_notify_event
```

```
[0053/0000] | MSG | 00:31:35.924 | SNS/Low | [sns_oem1_sensor_instance_island.c     73] accel data: x -1.070162, y -0.864270, z
9.638642
```
← ***accel data fed to OEM1 algorithm in sns_oem1_inst_notify_event()***

```
[0053/0000] | MSG | 00:31:35.924 | SNS/Low | [sns_oem1_sensor_instance_island.c     87] OEM1 output: x 100.000000, y -0.864270, z
9.638642
```
← ***Output of OEM1 algorithm calculated from accel data as per logic in sns_oem1_inst_notify_event()***

References

References

Acronyms	
Acronym or term	**Definition**
HAL	Hardware abstraction layer
SEE	Sensors Execution Environment
SLPI	Sensor Low Power Island
USTA	Unified Sensor Test Application

Questions?

https://createpoint.qti.qualcomm.com