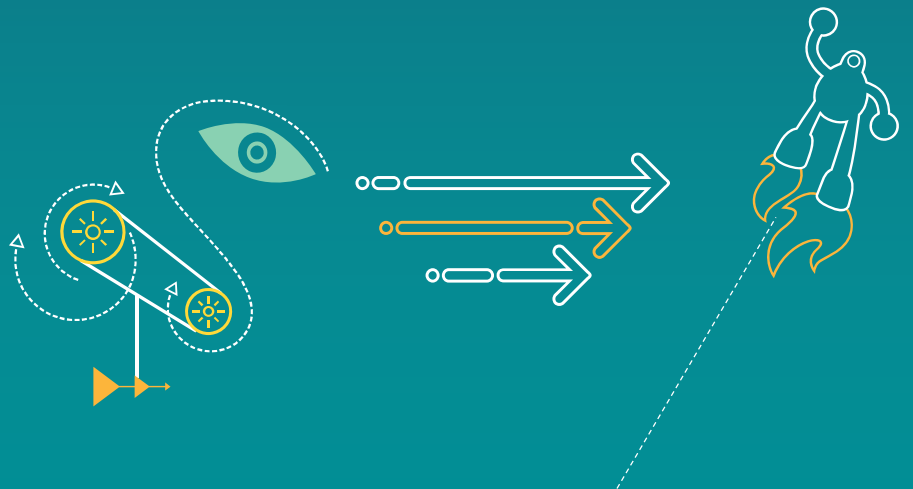
Adding a Custom Algorithm with SEE

Qualcomm Technologies, Inc.

80-P9301-67 Rev. A



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

Revision History

Revision	Date	Description
A	June 2017	Initial Release

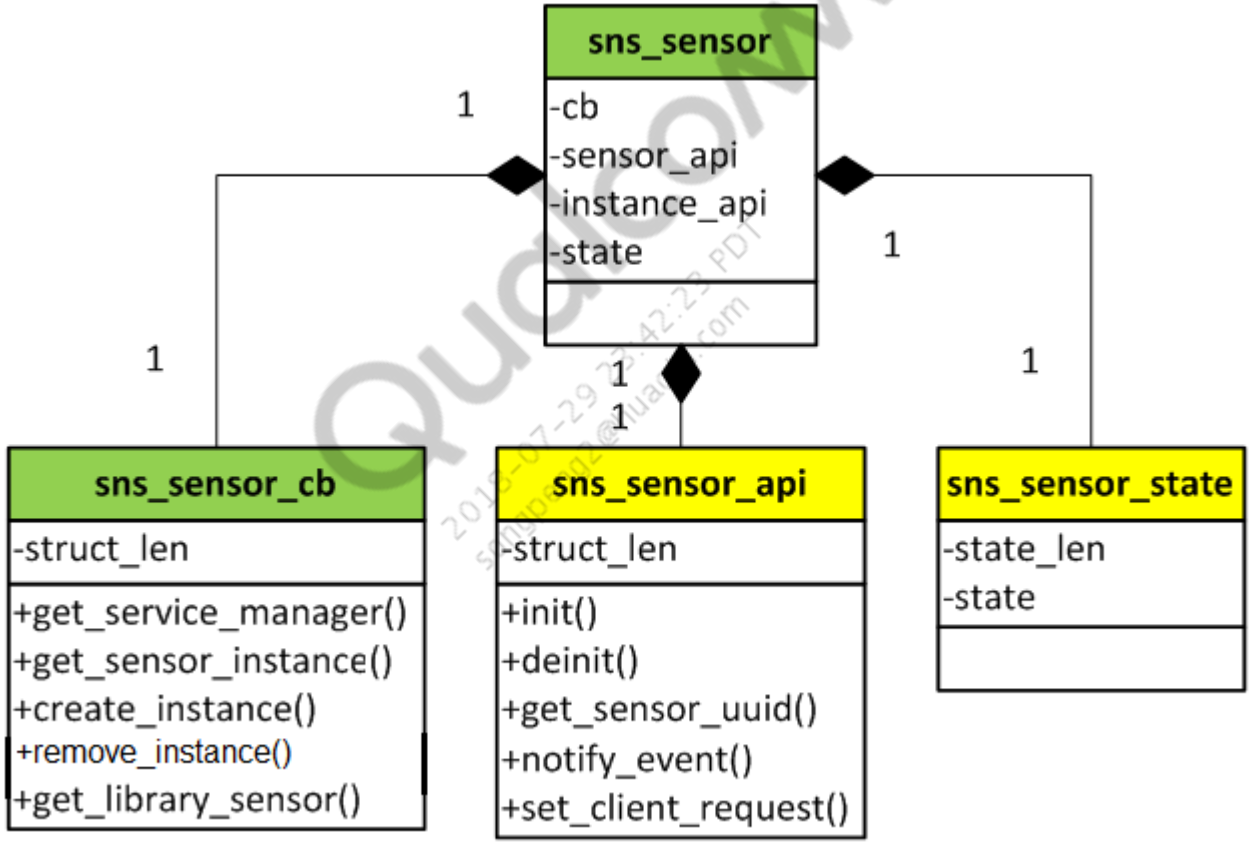
Agenda

Document Objectives

- Describes how customers add a custom algorithm with Sensors Execution Environment (SEE) on SDM845.
- Discusses OEM1, a template algorithm that is provided by Qualcomm as reference, and how to test it.
- See *SDM845 Sensors Overview* (80-P9301-34) for introductory information on SEE and relevant APIs. Some knowledge of sensor and sensor instance APIs is assumed.
  - 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

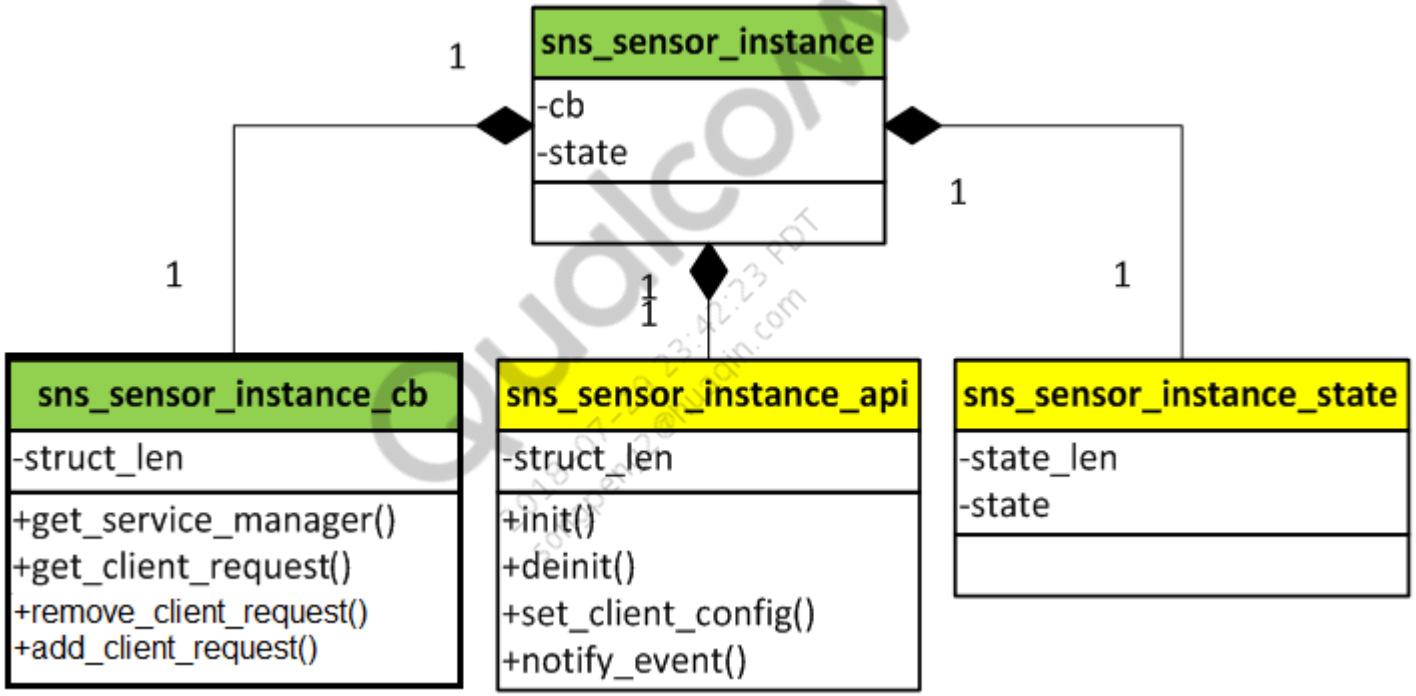
Sensor API



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.

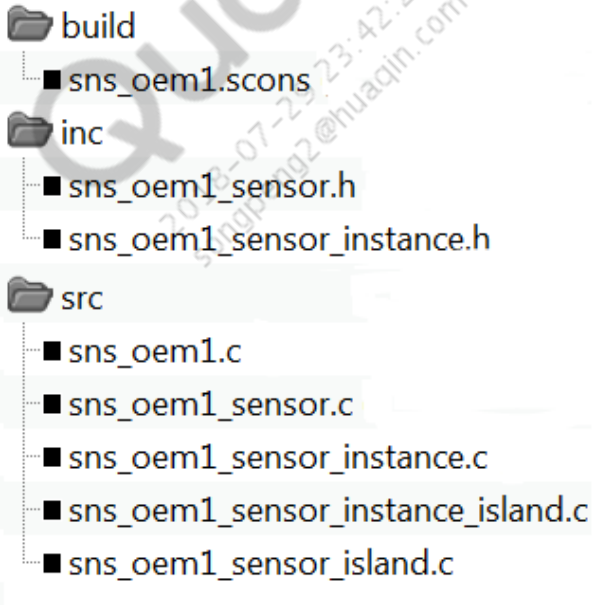
Algorithm Development

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 physical sensor drivers) reside under the slpi_proc\ssc\sensors\<algo_name> directory with SEE.
- OEM1 is provided as a sample algorithm in the default Qualcomm 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.

Defining a .proto File for Custom Algorithm (1 of 2)

- 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 (2 of 2)

- 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 (1 of 2)

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 (2 of 2)

Algorithm logic

- `sns_oem1_inst_notify_event()` in sns_oem1_sensor_instance_island.c

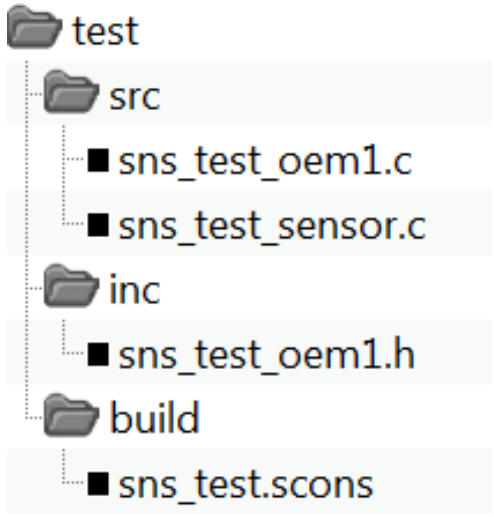
```
//This is dummy logic for OEM1 demonstration purposes
//OEMs can replace with their algo logic
if (0 < temp[2])
{
        accel_payload[0]=100;
        accel_payload[1]=temp[1];
        accel_payload[2]=temp[2];
}
else
{
        accel_payload[0]=9;
        accel_payload[1]=temp[1];
        accel_payload[2]=temp[2];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

- `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

Testing Custom Algorithm

OEM1 Test Sensor

- 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



Compiling SLPI with OEM1 and OEM1 Test Sensor

- Use the following build flags in the build command:
  - `USES_DELAYED_INIT` ← To view the boot-up time logs and OEM1 initialization logs
  - `SNS_TEST_OEM1` ← To enable test sensor to stream OEM1
- After HAL and application processor-side support is enabled, these two build flags are not required for end-to-end testing.

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

Title	Number
Qualcomm Technologies, Inc.	
SDM845 Sensors Overview	80-P9301-34

Acronym or term	Definition
HAL	Hardware abstraction layer
SEE	Sensors Execution Environment
SLPI	Sensor Low Power Island

Questions?

For additional information or to submit technical questions, go to https://createpoint.qti.qualcomm.com