



## **GENIVI SensorsService**

# **Component Specification**

Release 4.0.0-alpha

Status: Released

16.12.2015

**Accepted for release by:**

Approved by the GENIVI expert group Location Based Services (LBS) and the GENIVI system architecture team (SAT).

**Abstract:**

This document describes the API 4.0.0 of the **SensorsService** Abstract Component.

**Keywords:**

SensorsService, Sensors, Positioning API.

SPDX-License-Identifier: CC-BY-SA-4.0

Copyright © 2012, BMW Car IT GmbH, Continental Automotive GmbH, PCA Peugeot Citroën, XS Embedded GmbH

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, 444 Castro Street, Suite 900, Mountain View, California, 94041, USA.

# Table of Contents

Change History .....	5
1. Introduction .....	6
2. Terminology .....	7
3. Requirements .....	8
1. Requirements Diagram .....	8
AGPS Support .....	10
Forward a Set of Sensor Values .....	10
Provides Data via IPC .....	10
Support of different Global Navigation Satellite Systems (GNSS) to calculate the current position. ....	10
Accelerator Sensor .....	10
Access to Sensor Services .....	11
Car Configuration Data .....	11
Data Latency for GNSS and DR Signals .....	11
Enhanced Position .....	12
Extended Acceleration Sensor .....	12
Extended GNSS Service .....	12
Extended Gyroscope Sensor Service .....	13
GNSS Service .....	13
PPS Signal .....	13
Inclination Sensor .....	14
Odometer Sensor .....	14
ReverseGear Sensor .....	14
Sensor Directory .....	14
Sensor Meta-Data .....	15
Sensor Signal Timestamp .....	15
Signal Measurement Units .....	15
Signal Values Type Compatibility .....	16
Simple Gyroscope Sensor Service .....	16
Slip Angle Sensor .....	16
SteeringAngle Sensor .....	16
Vehicle State Sensor .....	17
VehicleSpeed Sensor .....	17
Wheel Tick/Speed Sensor Service .....	17
4. Architecture .....	18
1. SensorsService .....	18
2. SensorsService Diagram .....	18
3. Traceability Diagram .....	19
1. Context .....	20
2. Context Diagram .....	20



## Change History

Version	Date	Author	Change
0.1	27.08.2013	M. Residori	Document Created
0.2	18.11.2013	M. Residori	Document generated from the Enterprise Architect Model
0.3	27.03.2014	M. Residori	Added copyright notes
3.0.0-alpha	24.04.2014	M. Residori	Changed license version from 3.0 to 4.0
3.0.0-alpha	10.12.2014	M. Residori	Updated API description
3.0.0-alpha	20.01.2015	H. Schmidt	Fix copy/paste error
3.0.0	20.01.2015	M. Residori	Changed status to “Released” (after System Architecture Team approval)
3.0.1	01.04.2015	H. Schmidt	Bugfix in gyroscope/acceleration API
4.0.0.-alpha	16.12.2015	M. Residori	Updated API description

# **1. Introduction**

This document describes the API of the SensorsService component.

## 2. Terminology

<i>Term</i>	<i>Description</i>
GNSS	Global Navigation Satellite System

### **3. Requirements**

#### ***1. Requirements Diagram***

This diagram shows an overview of all requirements in the area of positioning.

The requirements are organized in four groups:

1. SW-POS: general requirements
2. SW-GNSS: requirements related to the GNSS receiver
3. SW-SNS: requirements related to the vehicle sensors
4. SW-ENP: requirements related to enhanced positioning



# req Requirements

## SW-POS

<b>Sensor Meta-Data</b> <i>tags</i> Originator = Thomas Himbacher (BMW) Priority = P1 Rationale =	<b>Sensor Signal Timestamp</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P1 Rationale =	<b>Signal Measurement Units</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P1 Rationale =	<b>Car Configuration Data</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =
<b>Signal Values Type Compatibility</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =	<b>Access to Sensor Services</b> <i>tags</i> Originator = Thomas Himbacher (BMW) Priority = P2 Rationale =	<b>Sensor Directory</b> <i>tags</i> Originator = Thomas Himbacher (BMW) Priority = P2 Rationale =	Support of different Global Navigation Satellite Systems (GNSS) to calculate the current position. <i>tags</i> Originator = Philippe Colliot (PSA) Priority = P2 Rationale = <memo>

## SW-POS-GNSS

<b>GNSS Service</b> <i>tags</i> Originator = Thomas Himbacher (BMW) Priority = P1 Rationale =	<b>Extended GNSS Service</b> <i>tags</i> Originator = Thomas Himbacher (BMW) Priority = P2 Rationale =	<b>PPS Signal</b> <i>tags</i> Originator = Carsten Iserl (BMW) Priority = P2 Rationale =	<b>AGPS Support</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P3 Rationale =
-----------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------

## SW-POS-SNS

<b>VehicleSpeed Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =	<b>Odometer Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =	<b>Wheel Tick/Speed Sensor Service</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =	<b>Simple Gyroscope Sensor Service</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =
<b>Extended Gyroscope Sensor Service</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P3 Rationale =	<b>ReverseGear Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =	<b>Accelerator Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =	<b>Extended Acceleration Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P3 Rationale =
<b>Inclination Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P3 Rationale =	<b>Slip Angle Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P3 Rationale =	<b>Vehicle State Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P3 Rationale =	<b>SteeringAngle Sensor</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P3 Rationale =

## SW-POS-ENP

<b>Enhanced Position</b> <i>tags</i> Originator = Thomas Himbacher (BMW) Priority = P1 Rationale =	<b>Provides Data via IPC</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P1 Rationale =	<b>Forward a Set of Sensor Values</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =	<b>Data Latency for GNSS and DR Signals</b> <i>tags</i> Originator = Thomas Bader (BMW) Priority = P2 Rationale =
----------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------

Figure: 1

AGPS Support		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The software platform provides the possibility to inject AGPS "Assisted GPS" data to the GPS device.		
<b>Rationale:</b> This allows to speed up the time to get a valid (fixed) GPS position.		

Forward a Set of Sensor Values		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The Enhanced Position contains in addition to the Position and Course values as well a set of sensor data. <ul style="list-style-type: none"> <li>- yawRate in degrees per second</li> <li>- filter status</li> <li>- accuracy information in form of sigma values for every direction [m] and the covariance between latitude and longitude in m<sup>2</sup>.</li> <li>- number of used, tracked and visible satellites.</li> </ul>		
<b>Rational:</b> Some clients (e.g. Map Matcher) needs the basic DR filtered position specific sensor values as additional input for the decision algorithm.		

Provides Data via IPC		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The enhanced position is accessible for multiple clients on the platform at the same time. An IPC is used to deliver to the clients the Enhanced Position data fields.		
<b>Rational:</b> Several SW components in the system are clients for the result of the filtered position and need to access the data.		

Support of different Global Navigation Satellite Systems (GNSS) to calculate the current position.		
«GFunctionalRequirement»	Priority: Medium	
The interfaces are defined in such a way that client applications don't need to know the details of the GNSS in use (e.g. GPS, Galileo, GLONASS, Compass).		

Accelerator Sensor		
«GFunctionalRequirement»	Priority: Medium	

**Description:**

The software platform provides a sensor, which delivers the vehicle acceleration in the driving direction (x Axis, see reference system). The sensor value is delivered in m/s<sup>2</sup>. Sensor value of temperature near the sensor is optional.

Configuration data about placement and orientation of the sensor can be provided optionally.

**Rational:**

Used for optimizing the dead reckoning solution.

### Access to Sensor Services

«GFunctionalRequirement» Priority: Medium

**Description:**

The software platform delivers signals to multiple client applications concurrently by the Sensor Service.

**Rational:**

This allows for multiple Client Applications to share a single Sensor.

### Car Configuration Data

«GFunctionalRequirement» Priority: Medium

**Description:**

The software platform provides car configuration data, that contains general vehicle details (e.g. physical dimensions of car, distance of axis, driven axis, etc).

Sensor related configuration data depends on the specific sensor requirements (e.g. position of sensor) and is included with the specific sensors.

- Position of center of gravity
- Position of front and rear axle
- driven axles
- seat count
- vehicle mass
- vehicle width
- track width

**Rational:**

DR module needs the detailed information for more accurate calculations.

### Data Latency for GNSS and DR Signals

«GNonFunctionalRequirement» Priority: Medium

**Description:**

The software platform provides the signals of the GNSS, Extended GNSS and enhanced position in less than 300 ms after acquisition.

**Rational:**

This guarantees that the tracked current position does not deviate much from the actual position.

Enhanced Position		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The software platform delivers the filtered (i.e. combined GNSS and vehicle sensor) position as the Enhanced Position, which is the result of the dead reckoning calculation. The Enhanced Position contains: <ul style="list-style-type: none"> <li>- Position expressed as WGS 84 longitude and latitude (unit is tenth of microdegree (degree x 10<sup>-7</sup>))</li> <li>- the Altitude 'above mean sea level' in meters (corrected by GeoID)</li> <li>- Heading in degrees relative to the true north</li> <li>- Climb</li> <li>- Speed in meters per seconds, positive in the forward direction</li> </ul> <b>Rational:</b> Other SW-components on the same platform want to access the improved GNSS position, which is calculated by a dead reckoning algorithm.		

Extended Acceleration Sensor		
«GFunctionalRequirement»	Priority: Low	
<b>Description:</b> The software platform provides a sensor, which provides the acceleration on the additional axis y (left-side) and z (up). The position of the sensor in 3D space in relation to the reference point is given. The angles of the sensor can be specified in the car configuration data. The standard deviations for the sensors can be specified for each axis. <b>Rational:</b> Used for optimizing the dead reckoning solution.		

Extended GNSS Service		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The software platform provides an extension to the GNSS Service with optional information. <p>Accuracy:</p> <ul style="list-style-type: none"> <li>- fixStatus</li> <li>- hdop, pdop, vdop</li> <li>- numberOfSatellites</li> <li>- sigmaLatitude, sigmaLongitude, sigmaAltitude</li> </ul> <p>Satellite Details:</p> <ul style="list-style-type: none"> <li>- Information per satellite: azimuth, elevation, inUse, SatelliteId, signalNoiseRatio</li> </ul> <p>Course Details:</p> <ul style="list-style-type: none"> <li>- speed for 3-axis</li> </ul> <p>Antenna:</p> <ul style="list-style-type: none"> <li>- Antenna Position in 3D coordinates in relation to the reference point (see reference system).</li> </ul> <p>Updated at least with 1Hz frequency additionally to the Signals provided by GNSS-Only Service.  The GNSS Service should provide the capability to switch between different GNSS-Devices (e.g. Galileo,</p>		

GPS, etc)

**Rational:**

These data are used for improved positioning based on GNSS.

### Extended Gyroscope Sensor Service

«GFunctionalRequirement» Priority: Low

**Description:**

The software platform includes the sensor that delivers

- pitch rate
- roll rate

This sensor values extend the simple gyroscope sensor.

Sign of is defined by rule of right hand (thumb direction: left and front, see reference system).

Car configuration data need to provide position angles according to vehicle reference system.

**Rational:**

This Sensor Service is used in Dead Reckoning calculations of the vehicle position.

### GNSS Service

«GFunctionalRequirement» Priority: High

**Description:**

The software platform includes a service that provides the following GNSS Signals updated at least with 1Hz frequency:

Position:

- position expressed as WGS 84 altitude, longitude and latitude in tenth of microdegree (degree x  $10^{-7}$ )

Course:

- speed in meters per second
- climb
- heading relative to true north expressed in degrees

Timestamp and date as UTC.

**Rational:**

These data are contained in NMEA 0183 \$GPGGA and \$GPRMC messages and provide the minimum information required for GNSS-only vehicle positioning.

### PPS Signal

«GFunctionalRequirement» Priority: Medium

**Description:**

1) For accurate timing the 1 PPS (pulse per second) signal from the GPS receiver is provided within the positioning framework.

The PPS is a hardware signal which is a UTC synchronized pulse.

The duration between the pulses is 1s +/- 40ns and the duration of the pulse is configurable (e.g. it could be 100ms or 200ms).

The pulses occur exactly at the UTC full second timeslots.

2) One option is to provide this signal in the positioning framework as an interrupt service routine and the difference to the system time can be accessed by a getter. This provides a synchronization of the system time to UTC.

**Rationale:**

Used for synchronizing the timing of the ECU.

## Inclination Sensor

«GFunctionalRequirement» Priority: Low

**Description:**

The software platform provides the inclination of the road in longitudinal direction, i.e. in the direction of movement [°]. Estimated gradient of the road in transverse direction [°]. In unstable driving situations this value might not be available.

**Rational:**

This Sensor is used for optimizations in Dead Reckoning calculations of the vehicle position.

## Odometer Sensor

«GFunctionalRequirement» Priority: Medium

**Description:**

The software platform includes a Sensor that delivers the traveled distance.  
Distance in [cm] with at least 5Hz as a running counter with overflow to support multiple clients.

**Rational:**

Odometer is sometimes the only speed related Signal available to the head unit.

## ReverseGear Sensor

«GFunctionalRequirement» Priority: Medium

**Description:**

The software platform includes a Sensor that delivers the information if the reverse gear is enabled or not.

**Rational:**

The direction of movement is included in the vehicle speed. This information is only used to detect reverse gear or not.

## Sensor Directory

«GFunctionalRequirement» Priority: Medium

**Description:**

Client Applications are able to query what Sensors are currently available.

**Rational:**

This allows for development of flexible applications that do not know what sensor data are available in the vehicle a priori. Client shall check first this directory to find out which ones are available; use meta-data to choose one of interest and use provided data to connect to necessary services.

Sensor Meta-Data		
«GFunctionalRequirement»	Priority: High	
<b>Description:</b> The software platform provides the following information about the Sensor and the related output Signals: <ul style="list-style-type: none"> <li>- Sensor Identifier that is unique within the system</li> <li>- Sensor Category (Physical/Logical)</li> <li>- Sensor Type (GPS, Odometer, Map Matching, etc.)</li> <li>- Sensor Sub-Type (ordinary GPS, differential GPS, etc.)</li> <li>- Output Signals (Longitude, Latitude, Course, Speed, etc.)</li> <li>- Output Signal Sampling Frequency (1 Hz, 10 Hz, irregular, etc.)</li> <li>- Output Signal Measurement Units (kilometers per hour; meters per second; etc.)</li> </ul>		
<b>Rational:</b> Sensor clients need that information in order to correctly handle data provided by sensor service and to adapt to the variation in the signal data delivery.		

Sensor Signal Timestamp		
«GFunctionalRequirement»	Priority: High	
<b>Description:</b> The software platform provides for each sample returned by the Sensor Service the timestamp, when it is accompanied. The timestamp corresponds to the time point of the sample acquisition or calculation. Timestamps are derived from the same clock that is accessible to the Client Applications. Timestamp is delivered with a accuracy of milliseconds.		
<b>Rational:</b> Measurement timestamps are important for proper functioning of most processing algorithms. For instance, algorithms for sensor calibration and dead reckoning typically use data from multiple sensors in conjunction, e.g. logical sensor.		

Signal Measurement Units		
«GFunctionalRequirement»	Priority: High	
<b>Description:</b> The software platform delivers signal values in universal, implementation independent units. It's preferred to use SI-units. For example, a gyroscope signal should be measured in millidegrees per second instead of A/D converter counts.		
<b>Rational:</b> This decouples the client applications from the implementation details of individual sensor devices.		

Signal Values Type Compatibility		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> All Sensor Services that provide Signals referring to the same physical quantity deliver their data in the same format (including API signatures, data type and measurement units). However, sampling frequency, accuracy etc. can differ.		
<b>Rational:</b> Sensor service clients are able to use multiple Sensor Services without changes in the interfaces.		

Simple Gyroscope Sensor Service		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The software platform includes the Sensor that delivers <ul style="list-style-type: none"> <li>- yaw rate: the rate of the vehicle heading change</li> <li>-temperature</li> <li>- status:(temperature compensated or not, etc)</li> </ul> at the frequency of at least 5Hz. Unit of yaw rate is "degrees per second". <p>Sign of yaw rate is defined by rule of right hand (thumb direction: up) (see reference system)</p>		
<b>Rational:</b> This Sensor Service is used in Dead Reckoning calculations of the vehicle position.		

Slip Angle Sensor		
«GFunctionalRequirement»	Priority: Low	
<b>Description:</b> Platform provides a sensor, which delivers the value slip angle in degrees [°]. It is defined as the angle between the fixed car axis (direction of driving) and the real direction of vehicle movement. The direction and sign is defined equal to the yaw rate (See reference system).		
<b>Rational:</b> This Sensor is used for optimizations in Dead Reckoning calculations of the vehicle position.		

SteeringAngle Sensor		
«GFunctionalRequirement»	Priority: Low	
<b>Description:</b> This sensor provides the angles of the front and rear wheels and the steering wheel in degrees. Configuration values can be provided for sigmas and steering ratio.		
<b>Rational:</b> Is used as additional element for plausibilisation of the yaw rate in the dead reckoning module.		



Vehicle State Sensor		
«GFunctionalRequirement»	Priority: Low	
<b>Description:</b> The software platform provides a sensor, giving the state of certain vehicle systems: ABS: on/off ESP: on/off ASC: on/off (stability control) breaks: on/off  <b>Rational:</b> This Sensor is used for optimizations in Dead Reckoning calculations of the vehicle position.		

VehicleSpeed Sensor		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The software platform includes a Sensor that delivers the vehicle speed. Filtered vehicle speed in [m/s] with a frequency of at least 5Hz. Direction is given by the sign of this value.  <b>Rational:</b> Vehicle speed is sometimes the only speed related signal available to the head unit.		

Wheel Tick/Speed Sensor Service		
«GFunctionalRequirement»	Priority: Medium	
<b>Description:</b> The software platform provides a Sensor that delivers the running counter of partial wheel revolutions at the frequency of at least 5Hz or the already calculated wheelspeed (speed in [m/s] or angular speed).  The resolution of a single wheel revolution (i.e. the number of ticks per revolution) is included with the Sensor Service meta-data.  This identifiers specify the wheel of measurement: 0: Average of non driven axle 1: Left front wheel 2: Right front wheel 3: Left rear wheel 4: Right rear wheel Unit: [ticks].  <b>Rational:</b> This Sensor typically registers ‘ticks’ from a wheel, adds them up and sends to the vehicle bus with a certain interval. The number of ‘ticks’ per complete wheel revolution is known in advance. In some cases, the data from multiple wheels are averaged. Other implementations send the already precalculated speed per wheel or axle, which is a valid replacement for most use cases.		

## 4. Architecture

### 1. *SensorsService*

The SensorsService is a component that is responsible for retrieving data from the available vehicle sensors and making them available to other client applications. It hides dependencies to hardware and IPC mechanism.

In systems that implement the EnhancedPositionService component, the SensorsService is typically implemented as a C library that is dynamically linked by the EnhancedPositionService.

### 2. *SensorsService Diagram*

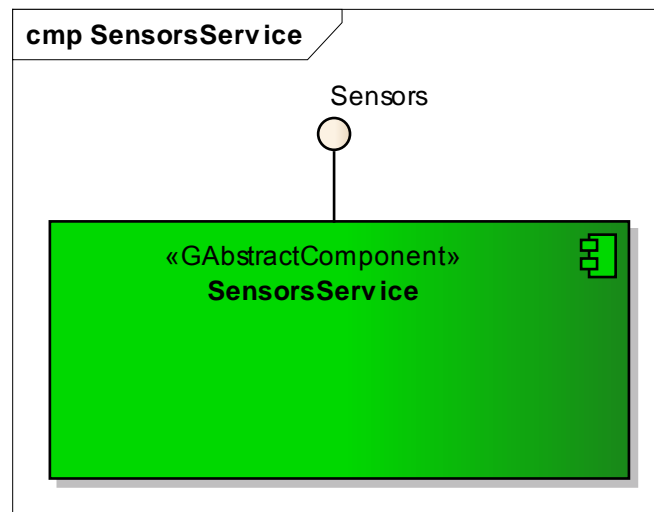


Figure: 2

### 3. Traceability Diagram

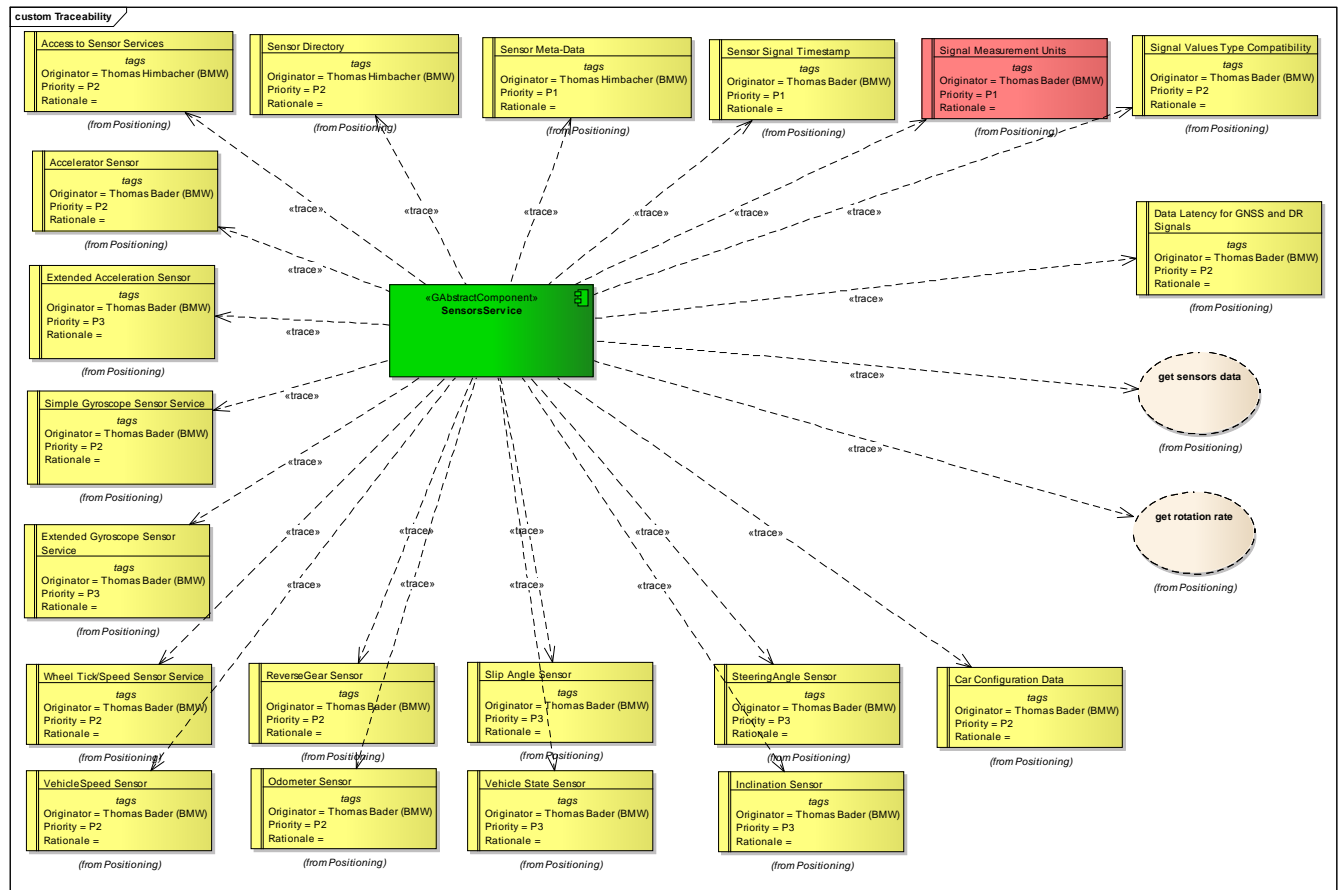


Figure: 3

## 1. Context

This diagram shows how the SensorsService interacts with its client application: the EnhancedPositionService.

## 2. Context Diagram

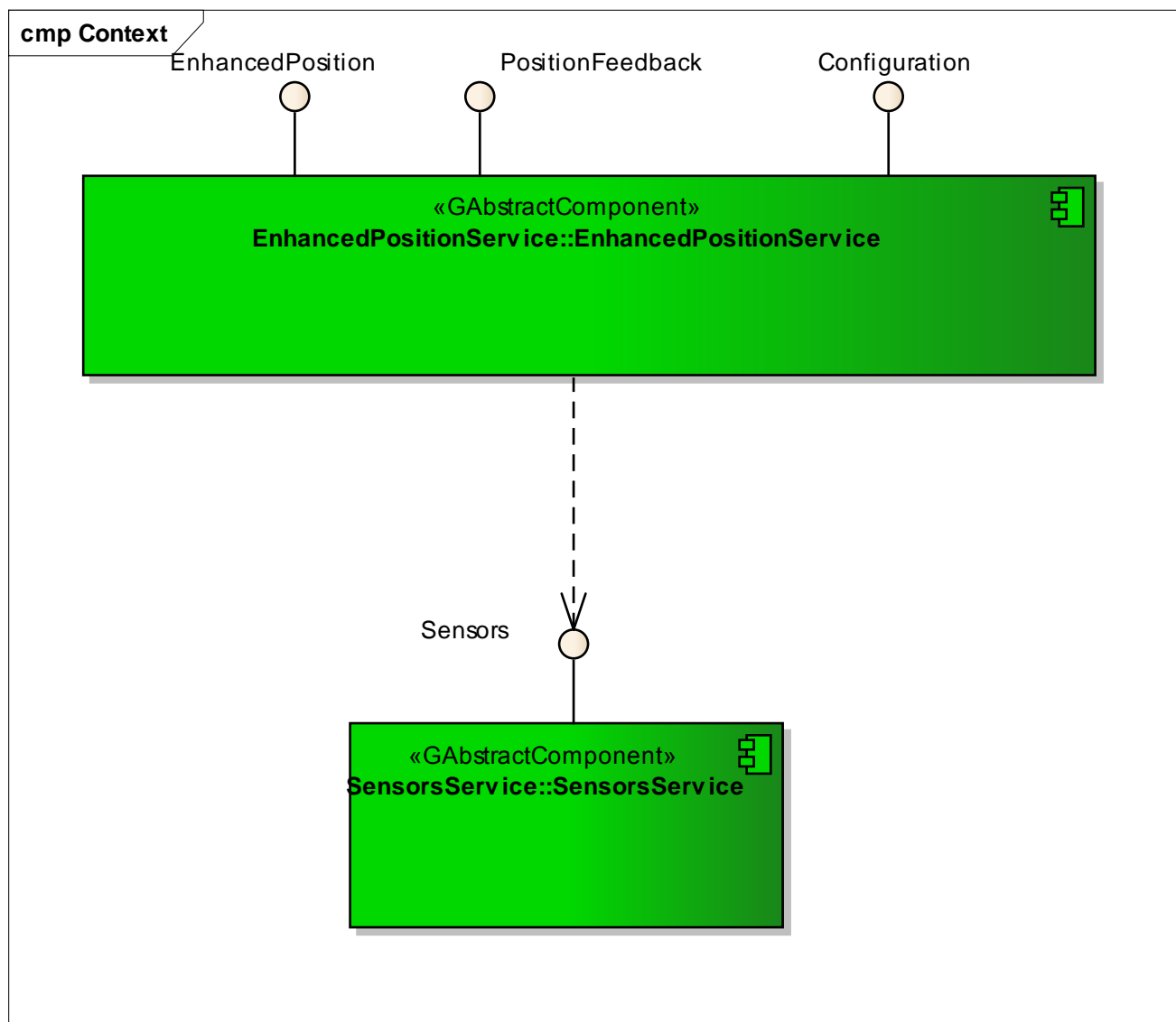


Figure: 4

## SensorsService

Generated by Doxygen 1.8.6

Wed Dec 2 2015 14:01:09

## Contents

<b>1</b>	<b>Class Documentation</b>	<b>1</b>
1.1	TAccelerationConfiguration Struct Reference	1
1.1.1	Detailed Description	2
1.1.2	Member Data Documentation	3
1.2	TAccelerationData Struct Reference	3
1.2.1	Detailed Description	4
1.2.2	Member Data Documentation	4
1.3	TDistance3D Struct Reference	4
1.3.1	Detailed Description	5
1.3.2	Member Data Documentation	5
1.4	TGyroscopeConfiguration Struct Reference	5
1.4.1	Detailed Description	6
1.4.2	Member Data Documentation	6
1.5	TGyroscopeData Struct Reference	7
1.5.1	Detailed Description	7
1.5.2	Member Data Documentation	7
1.6	TInclinationData Struct Reference	8
1.6.1	Detailed Description	8
1.6.2	Member Data Documentation	8
1.7	TOdometerData Struct Reference	9
1.7.1	Detailed Description	9
1.7.2	Member Data Documentation	9
1.8	TReverseGearData Struct Reference	10
1.8.1	Detailed Description	10
1.8.2	Member Data Documentation	10
1.9	TSensorMetaData Struct Reference	10
1.9.1	Detailed Description	10
1.9.2	Member Data Documentation	11
1.10	TSensorStatus Struct Reference	11
1.10.1	Detailed Description	11
1.10.2	Member Data Documentation	11
1.11	TSlipAngleData Struct Reference	12
1.11.1	Detailed Description	12
1.11.2	Member Data Documentation	12
1.12	TSteeringAngleConfiguration Struct Reference	12
1.12.1	Detailed Description	12
1.12.2	Member Data Documentation	13
1.13	TSteeringAngleData Struct Reference	13

1.13.1 Detailed Description . . . . .	13
1.13.2 Member Data Documentation . . . . .	13
1.14 TVehicleDataConfiguration Struct Reference . . . . .	14
1.14.1 Detailed Description . . . . .	14
1.14.2 Member Data Documentation . . . . .	14
1.15 TVehicleSpeedData Struct Reference . . . . .	15
1.15.1 Detailed Description . . . . .	15
1.15.2 Member Data Documentation . . . . .	15
1.16 TVehicleStateData Struct Reference . . . . .	16
1.16.1 Detailed Description . . . . .	16
1.16.2 Member Data Documentation . . . . .	16
1.17 TWheelConfiguration Struct Reference . . . . .	17
1.17.1 Detailed Description . . . . .	17
1.17.2 Member Data Documentation . . . . .	17
1.18 TWheelData Struct Reference . . . . .	18
1.18.1 Detailed Description . . . . .	18
1.18.2 Member Data Documentation . . . . .	18
<b>2 File Documentation</b>	<b>19</b>
2.1 acceleration.h File Reference . . . . .	19
2.1.1 Typedef Documentation . . . . .	20
2.1.2 Enumeration Type Documentation . . . . .	20
2.1.3 Function Documentation . . . . .	21
2.2 gyroscope.h File Reference . . . . .	24
2.2.1 Typedef Documentation . . . . .	25
2.2.2 Enumeration Type Documentation . . . . .	25
2.2.3 Function Documentation . . . . .	26
2.3 inclination.h File Reference . . . . .	29
2.3.1 Typedef Documentation . . . . .	29
2.3.2 Enumeration Type Documentation . . . . .	30
2.3.3 Function Documentation . . . . .	30
2.4 odometer.h File Reference . . . . .	32
2.4.1 Typedef Documentation . . . . .	33
2.4.2 Enumeration Type Documentation . . . . .	33
2.4.3 Function Documentation . . . . .	33
2.5 reverse-gear.h File Reference . . . . .	36
2.5.1 Typedef Documentation . . . . .	36
2.5.2 Enumeration Type Documentation . . . . .	37
2.5.3 Function Documentation . . . . .	37
2.6 slip-angle.h File Reference . . . . .	39

2.6.1	Typedef Documentation . . . . .	39
2.6.2	Enumeration Type Documentation . . . . .	40
2.6.3	Function Documentation . . . . .	40
2.7	sns-init.h File Reference . . . . .	42
2.7.1	Macro Definition Documentation . . . . .	42
2.7.2	Function Documentation . . . . .	42
2.8	sns-meta-data.h File Reference . . . . .	43
2.8.1	Enumeration Type Documentation . . . . .	43
2.8.2	Function Documentation . . . . .	44
2.9	sns-status.h File Reference . . . . .	44
2.9.1	Typedef Documentation . . . . .	45
2.9.2	Enumeration Type Documentation . . . . .	46
2.10	steering-angle.h File Reference . . . . .	46
2.10.1	Typedef Documentation . . . . .	47
2.10.2	Enumeration Type Documentation . . . . .	47
2.10.3	Function Documentation . . . . .	47
2.11	vehicle-data.h File Reference . . . . .	50
2.11.1	Enumeration Type Documentation . . . . .	51
2.11.2	Function Documentation . . . . .	52
2.12	vehicle-speed.h File Reference . . . . .	52
2.12.1	Typedef Documentation . . . . .	53
2.12.2	Enumeration Type Documentation . . . . .	53
2.12.3	Function Documentation . . . . .	53
2.13	vehicle-state.h File Reference . . . . .	55
2.13.1	Typedef Documentation . . . . .	56
2.13.2	Enumeration Type Documentation . . . . .	56
2.13.3	Function Documentation . . . . .	56
2.14	wheel.h File Reference . . . . .	58
2.14.1	Typedef Documentation . . . . .	59
2.14.2	Enumeration Type Documentation . . . . .	60
2.14.3	Function Documentation . . . . .	62
	<b>Index</b>	<b>65</b>

## 1 Class Documentation

### 1.1 TAccelerationConfiguration Struct Reference

```
#include <acceleration.h>
```



## Public Attributes

- float [dist2RefPointX](#)
- float [dist2RefPointY](#)
- float [dist2RefPointZ](#)
- float [angleYaw](#)
- float [anglePitch](#)
- float [angleRoll](#)
- float [sigmaX](#)
- float [sigmaY](#)
- float [sigmaZ](#)
- uint32\_t [typeBits](#)
- uint32\_t [validityBits](#)

### 1.1.1 Detailed Description

Static configuration data for the acceleration sensor service.

BEGIN Explanation of the angleYaw, anglePitch angleRoll parameters The orientation of the accelerometer hardware (Xa, Ya, Za) with respect to the vehicle axis system (Xv, Yv, Zv) can be described using the angles (angleYaw, anglePitch, angleRoll) following the approach defined in ISO 8855:2011, section 5.2, table 1 Apply 3 rotations on the vehicle axis system until it matches the accelerometer axis system The rotation sequence is as follows

- first rotate by angleYaw about the Zv axis
- second rotate by anglePitch about the new (intermediate) Y axis
- third rotate by angleRoll about the new X axis Notes

the angles are frequently called "Euler angles" and the rotations "Euler rotations"

- a different order of the rotations would lead to a different orientation
- as the vehicle axis system is right-handed, also the accelerometer axis system must be right-handed

The vehicle axis system as defined in ISO 8855:2011(E). In this system, the axes (Xv, Yv, Zv) are oriented as follows

- Xv is in the horizontal plane, pointing forwards
- Yv is in the horizontal plane, pointing to the left
- Zv is perpendicular to the horizontal plane, pointing upwards For an illustration, see <https://collab.-genivi.org/wiki/display/genivi/LBSSensorServiceRequirementsBorg#LBSSensorServiceRequirementsBorg-ReferenceSystem>

When the accelerometer axes are not aligned with the vehicle axes, i.e. if any of the angles (angleYaw, anglePitch, angleRoll) is not zero then the raw measurement values of the accelerometer X, Y, Z axes may have to be transformed to the vehicle axis system by the client of this interface, depending on the type of application. Raw measurements are provided instead of already transformed values, because

- for accelerometers with less than 3 axes, the transformation is mathematically not well-defined
- some types of calibration operations are better performed on raw data

Implementors hint: The mathematics of this kind of transformation, like the derivation of the rotation matrixes is described in literature on strapdown navigation E.g. "Strapdown Inertial Navigation Technology", 2nd Edition by David Titterton and John Weston, section 3.6 END Explanation of the angleYaw, anglePitch angleRoll parameters

### 1.1.2 Member Data Documentation

#### 1.1.2.1 float TAccelerationConfiguration::anglePitch

Euler angle of second rotation, around pitch axis, to describe acceleration sensor orientation [degree]. For details, see above.

#### 1.1.2.2 float TAccelerationConfiguration::angleRoll

Euler angle of third rotation, around roll axis, to describe acceleration sensor orientation [degree]. For details, see above.

#### 1.1.2.3 float TAccelerationConfiguration::angleYaw

Euler angle of first rotation, around yaw axis, to describe acceleration sensor orientation [degree]. For details, see above.

#### 1.1.2.4 float TAccelerationConfiguration::dist2RefPointX

Distance of acceleration sensor from vehicle reference point (x-coordinate) [m].

#### 1.1.2.5 float TAccelerationConfiguration::dist2RefPointY

Distance of acceleration sensor from vehicle reference point (y-coordinate) [m].

#### 1.1.2.6 float TAccelerationConfiguration::dist2RefPointZ

Distance of acceleration sensor from vehicle reference point (z-coordinate) [m].

#### 1.1.2.7 float TAccelerationConfiguration::sigmaX

Standard error estimate of the x-acceleration [m/s<sup>2</sup>].

#### 1.1.2.8 float TAccelerationConfiguration::sigmaY

Standard error estimate of the y-acceleration [m/s<sup>2</sup>].

#### 1.1.2.9 float TAccelerationConfiguration::sigmaZ

Standard error estimate of the z-acceleration [m/s<sup>2</sup>].

#### 1.1.2.10 uint32\_t TAccelerationConfiguration::typeBits

Bit mask indicating the type of the used accelerometer. [bitwise or'ed [EAccelerationTypeBits](#) values].

#### 1.1.2.11 uint32\_t TAccelerationConfiguration::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EAccelerationConfigValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [acceleration.h](#)

## 1.2 TAccelerationData Struct Reference

```
#include <acceleration.h>
```

## Public Attributes

- `uint64_t timestamp`
- `float x`
- `float y`
- `float z`
- `float temperature`
- `uint32_t validityBits`

### 1.2.1 Detailed Description

The `AccelerationData` delivers the sensor values of the accelerometer. The coordinate system is the axis system of the accelerometer sensor, i.e. the x, y, z values are raw measurements without any conversion except probably averaging of multiple sensor readings over the measurement interval.

#### See Also

[TAccelerationConfiguration](#) for an explanation how to convert these values to the vehicle axis system

It is possible that not all values are populated, e.g. when only a 1-axis accelerometer is used. You must check the valid bits before usage.

### 1.2.2 Member Data Documentation

#### 1.2.2.1 `float TAccelerationData::temperature`

Temperature reading of the accelerometer sensor. If available it can be used for temperature compensation. The measurement unit is unspecified. Degrees celsius are preferred but any value linearly dependent on the temperature is fine.

#### 1.2.2.2 `uint64_t TAccelerationData::timestamp`

Timestamp of the acquisition of the accelerometer signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

#### 1.2.2.3 `uint32_t TAccelerationData::validityBits`

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EAccelerationValidityBits](#) values]. Must be checked before usage.

#### 1.2.2.4 `float TAccelerationData::x`

The acceleration in direction of the X-axis of the accelerometer sensor [ $\text{m/s}^2$ ].

#### 1.2.2.5 `float TAccelerationData::y`

The acceleration in direction of the Y-axis of the accelerometer sensor [ $\text{m/s}^2$ ].

#### 1.2.2.6 `float TAccelerationData::z`

The acceleration in direction of the Z-axis of the accelerometer sensor [ $\text{m/s}^2$ ].

The documentation for this struct was generated from the following file:

- [acceleration.h](#)

## 1.3 TDistance3D Struct Reference

```
#include <vehicle-data.h>
```

#### Public Attributes

- float [x](#)
- float [y](#)
- float [z](#)

#### 1.3.1 Detailed Description

3 dimensional distance used for description of geometric descriptions within the vehicle reference system.

The vehicle axis system as defined in ISO 8855:2011(E). In this system, the axes (Xv, Yv, Zv) are oriented as follows

- Xv is in the horizontal plane, pointing forwards
- Yv is in the horizontal plane, pointing to the left
- Zv is perpendicular to the horizontal plane, pointing upwards For an illustration, see <https://collab.-genivi.org/wiki/display/genivi/LBSSensorServiceRequirementsBorg#LBSSensorServiceRequirementsBorg-ReferenceSystem>

The reference point of the vehicle lies underneath the center of the rear axle on the surface of the road.

#### 1.3.2 Member Data Documentation

##### 1.3.2.1 float TDistance3D::x

Distance in x direction in [m] according to the reference coordinate system.

##### 1.3.2.2 float TDistance3D::y

Distance in y direction in [m] according to the reference coordinate system.

##### 1.3.2.3 float TDistance3D::z

Distance in z direction in [m] according to the reference coordinate system.

The documentation for this struct was generated from the following file:

- [vehicle-data.h](#)

## 1.4 TGyroscopeConfiguration Struct Reference

```
#include <gyroscope.h>
```

#### Public Attributes

- float [angleYaw](#)
- float [anglePitch](#)
- float [angleRoll](#)
- float [momentOfYawInertia](#)
- float [sigmaGyroscope](#)
- uint32\_t [typeBits](#)
- uint32\_t [validityBits](#)

### 1.4.1 Detailed Description

Static configuration data for the gyroscope sensor service.

BEGIN Explanation of the angleYaw, anglePitch angleRoll parameters The orientation of the gyroscope hardware (Xg, Yg, Zg) with respect to the vehicle axis system (Xv, Yv, Zv) can be described using the angles (angleYaw, anglePitch, angleRoll) following the approach defined in ISO 8855:2011, section 5.2, table 1 Apply 3 rotations on the vehicle axis system until it matches the gyroscope axis system The rotation sequence is as follows

- first rotate by angleYaw about the Zv axis
- second rotate by anglePitch about the new (intermediate) Y axis
- third rotate by angleRoll about the new X axis Notes

the angles are frequently called "Euler angles" and the rotations "Euler rotations"

- a different order of the rotations would lead to a different orientation
- as the vehicle axis system is right-handed, also the gyroscope axis system must be right-handed

The vehicle axis system as defined in ISO 8855:2011(E). In this system, the axes (Xv, Yv, Zv) are oriented as follows

- Xv is in the horizontal plane, pointing forwards
- Yv is in the horizontal plane, pointing to the left
- Zv is perpendicular to the horizontal plane, pointing upwards For an illustration, see <https://collab.-genivi.org/wiki/display/genivi/LBSSensorServiceRequirementsBorg#LBS-SensorServiceRequirementsBorg-ReferenceSystem>

When the gyroscope axes are not aligned with the vehicle axes, i.e. if any of the angles (angleYaw, anglePitch, angleRoll) is not zero then the raw measurement values of the gyroscope Z, Y, X axes may have to be transformed to the vehicle axis system by the client of this interface, depending on the type of application. Raw measurements are provided instead of already transformed values, because

- for gyroscopes with less than 3 axes, the transformation is mathematically not well-defined
- some types of calibration operations are better performed on raw data

Implementors hint: The mathematics of this kind of transformation, like the derivation of the rotation matrixes is described in literature on strapdown navigation E.g. "Strapdown Inertial Navigation Technology", 2nd Edition by David Titterton and John Weston, section 3.6 END Explanation of the angleYaw, anglePitch angleRoll parameters

### 1.4.2 Member Data Documentation

#### 1.4.2.1 float TGyroscopeConfiguration::anglePitch

Euler angle of second rotation, around pitch axis, to describe gyroscope orientation [degree]. For details, see above.

#### 1.4.2.2 float TGyroscopeConfiguration::angleRoll

Euler angle of third rotation, around roll axis, to describe gyroscope orientation [degree]. For details, see above.

#### 1.4.2.3 float TGyroscopeConfiguration::angleYaw

Euler angle of first rotation, around yaw axis, to describe gyroscope orientation [degree]. For details, see above.

#### 1.4.2.4 float TGyroscopeConfiguration::momentOfYawInertia

Moment of yaw inertia. The pitch and roll inertia moments are not provided as they are not relevant for positioning. [kg\*m<sup>2</sup>]

#### 1.4.2.5 float TGyroscopeConfiguration::sigmaGyroscope

Standard error estimate of the gyroscope for all directions. [degree/s]

#### 1.4.2.6 uint32\_t TGyroscopeConfiguration::typeBits

Bit mask indicating the type of the used gyroscope. [bitwise or'ed [EGyroscopeTypeBits](#) values].

#### 1.4.2.7 uint32\_t TGyroscopeConfiguration::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EGyroscopeConfigValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [gyroscope.h](#)

## 1.5 TGyroscopeData Struct Reference

```
#include <gyroscope.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- float [yawRate](#)
- float [pitchRate](#)
- float [rollRate](#)
- float [temperature](#)
- uint32\_t [validityBits](#)

### 1.5.1 Detailed Description

The GyroscopeData delivers the sensor values of the gyroscope. The coordinate system is the axis system of the gyroscope sensor, i.e. the yawRate, pitchRate, rollRate values are raw measurements without any conversion except probably averaging of multiple sensor readings over the measurement interval.

### See Also

[TGyroscopeConfiguration](#) for an explanation how to convert these values to the vehicle axis system

It is possible that not all values are populated, e.g. when only a 1-axis gyroscope is used. You must check the valid bits before usage.

### 1.5.2 Member Data Documentation

#### 1.5.2.1 float TGyroscopeData::pitchRate

Current angular rate measurement around the y/pitch-axis of the gyroscope sensor [degree/s]. Value range -100 / +100 degree/s. Frequency of at least 5Hz. Preferably 50Hz. A rotation front down is indicated by a positive sign.

### 1.5.2.2 float TGyroscopeData::rollRate

Current angular rate measurement around the x/roll-axis of the gyroscope sensor [degree/s]. Value range -100 / +100 degree/s. Frequency of at least 5Hz. Preferably 50Hz. A rotation right down is indicated by a positive sign.

### 1.5.2.3 float TGyroscopeData::temperature

Temperature reading of the gyroscope sensor. If available it can be used for temperature compensation. The measurement unit is unspecified. Degrees celsius are preferred but any value linearly dependent on the temperature is fine.

### 1.5.2.4 uint64\_t TGyroscopeData::timestamp

Timestamp of the acquisition of the gyroscope signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

### 1.5.2.5 uint32\_t TGyroscopeData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EGyroscopeValidityBits](#) values]. Must be checked before usage.

### 1.5.2.6 float TGyroscopeData::yawRate

Current angular rate measurement around the z/yaw-axis of the gyroscope sensor [degree/s]. Value range -100 / +100 degree/s. Frequency of at least 5Hz. Preferably 50Hz. A rotation to the left is indicated by a positive sign.

The documentation for this struct was generated from the following file:

- [gyroscope.h](#)

## 1.6 TInclinationData Struct Reference

```
#include <inclination.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- float [longitudinalGradientRoadway](#)
- float [traverseGradientRoadway](#)
- [EInclinationSensorStatus](#) [status](#)
- uint32\_t [validityBits](#)

### 1.6.1 Detailed Description

Inclination sensor service provides the inclination values.

### 1.6.2 Member Data Documentation

#### 1.6.2.1 float TInclinationData::longitudinalGradientRoadway

The inclination of the road in longitudinal direction, i.e. in the direction of movement [degree]. In instable driving situations this value might not be available.

#### 1.6.2.2 EInclinationSensorStatus TInclinationData::status

Status of the signals.

### 1.6.2.3 uint64\_t TIInclinationData::timestamp

Timestamp of the acquisition of the inclination signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

### 1.6.2.4 float TIInclinationData::traverseGradientRoadway

Estimated gradient of the road in transverse direction [degree]. In instable driving situations this value might not be available.

### 1.6.2.5 uint32\_t TIInclinationData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EIInclinationValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [inclination.h](#)

## 1.7 TOdometerData Struct Reference

```
#include <odometer.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- uint16\_t [travelledDistance](#)
- uint32\_t [validityBits](#)

### 1.7.1 Detailed Description

Odometer sensor service provides the travelled distance.

### 1.7.2 Member Data Documentation

#### 1.7.2.1 uint64\_t TOdometerData::timestamp

Timestamp of the acquisition of the odometer signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

#### 1.7.2.2 uint16\_t TOdometerData::travelledDistance

Distance in [cm] with at least 5Hz. Implemented as a running counter with overflow to support multiple clients and getter methods. As the representation of this value is done using 16 Bits the value can provide distances up 32767cm or 327.67m before overflowing.

#### 1.7.2.3 uint32\_t TOdometerData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EOdometerValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [odometer.h](#)



## 1.8 TReverseGearData Struct Reference

```
#include <reverse-gear.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- bool [isReverseGear](#)
- uint32\_t [validityBits](#)

### 1.8.1 Detailed Description

Reverse gear sensor service provides the current status of the reverse gear of the vehicle. This information is explicitly restricted to provide only the information if the reverse gear is engaged. The direction of movement is provided by the direction of the vehicle speed.

### 1.8.2 Member Data Documentation

#### 1.8.2.1 bool TReverseGearData::isReverseGear

True if the reverse gear is currently used. False otherwise.

#### 1.8.2.2 uint64\_t TReverseGearData::timestamp

Timestamp of the acquisition of the reverse gear signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

#### 1.8.2.3 uint32\_t TReverseGearData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EReverseGearValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [reverse-gear.h](#)

## 1.9 TSensorMetaData Struct Reference

```
#include <sns-meta-data.h>
```

### Public Attributes

- uint32\_t [version](#)
- [ESensorCategory](#) [category](#)
- [ESensorType](#) [type](#)
- uint32\_t [cycleTime](#)

### 1.9.1 Detailed Description

The software platform provides the following information about the Sensor and the related output Signals. Sensor clients need the meta data information in order to correctly handle data provided by sensor service and to adapt to the variation in the signal data delivery.

## 1.9.2 Member Data Documentation

### 1.9.2.1 ESensorCategory TSensorMetaData::category

Sensor Category (Physical/Logical).

### 1.9.2.2 uint32\_t TSensorMetaData::cycleTime

Sensor cycle time (update interval) in ms. 0 for irregular updates

### 1.9.2.3 ESensorType TSensorMetaData::type

Sensor Type (Odometer, Gyroscope, etc.).

### 1.9.2.4 uint32\_t TSensorMetaData::version

Version of the sensor service.

The documentation for this struct was generated from the following file:

- [sns-meta-data.h](#)

## 1.10 TSensorStatus Struct Reference

```
#include <sns-status.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- [ESensorStatus](#) [status](#)
- uint32\_t [validityBits](#)

### 1.10.1 Detailed Description

Container for sensor status information

## 1.10.2 Member Data Documentation

### 1.10.2.1 ESensorStatus TSensorStatus::status

Status of the sensor

### 1.10.2.2 uint64\_t TSensorStatus::timestamp

Timestamp of the sensor status transition [ms]. All sensor/GNSS timestamps must be based on the same time source.

### 1.10.2.3 uint32\_t TSensorStatus::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [ESensorStatusValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [sns-status.h](#)

## 1.11 TSlipAngleData Struct Reference

```
#include <slip-angle.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- float [slipAngle](#)
- uint32\_t [validityBits](#)

### 1.11.1 Detailed Description

Slip angle sensor service provides the slip angle value. ISO 8855:2011, section 5.2.9 "vehicle sideslip angle" The reference coordinate system for the slip angle is defined in ISO 8855:2011

You must check the valid bit(s) before usage.

### 1.11.2 Member Data Documentation

#### 1.11.2.1 float TSlipAngleData::slipAngle

Delivers the value slip angle in [degrees]. It is defined as the angle between the fixed car axis (direction of driving) and the real direction of vehicle movement. The direction and sign is defined equal to the yaw rate.

#### 1.11.2.2 uint64\_t TSlipAngleData::timestamp

Timestamp of the acquisition of the slip angle signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

#### 1.11.2.3 uint32\_t TSlipAngleData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [ESlipAngleValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [slip-angle.h](#)

## 1.12 TSteeringAngleConfiguration Struct Reference

```
#include <steering-angle.h>
```

### Public Attributes

- float [sigmaSteeringAngle](#)
- float [sigmaSteeringWheelAngle](#)
- float [steeringRatio](#)

### 1.12.1 Detailed Description

The SteeringAngleConfiguration delivers the static configuration values of the steering wheel sensor service.

### 1.12.2 Member Data Documentation

#### 1.12.2.1 float TSteeringAngleConfiguration::sigmaSteeringAngle

Standard error estimate of the front steer angle in [degree]. -1 if invalid.

#### 1.12.2.2 float TSteeringAngleConfiguration::sigmaSteeringWheelAngle

Standard error estimate of the steering wheel angle in [degree]. -1 if invalid.

#### 1.12.2.3 float TSteeringAngleConfiguration::steeringRatio

Ratio between steering wheel angle change and front steer angle change. ISO 8855:2011, section 7.1.13. Only valid when static. 0 if invalid. Unit: [-]

The documentation for this struct was generated from the following file:

- [steering-angle.h](#)

## 1.13 TSteeringAngleData Struct Reference

```
#include <steering-angle.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- float [front](#)
- float [rear](#)
- float [steeringWheel](#)
- uint32\_t [validityBits](#)

### 1.13.1 Detailed Description

The SteeringAngle delivers the sensor values of the steering angle. The reference coordinate system including the sign of the angles are defined in ISO 8855:2011, section 7

You must check the valid bits before usage.

### 1.13.2 Member Data Documentation

#### 1.13.2.1 float TSteeringAngleData::front

Returns the mean steer angle of the front wheels [degree]. ISO 8855:2011, section 7.1.3.

#### 1.13.2.2 float TSteeringAngleData::rear

Returns the mean steer angle of the rear wheels [degree]. ISO 8855:2011, section 7.1.3.

#### 1.13.2.3 float TSteeringAngleData::steeringWheel

Returns the angle of the steering wheel [degree]. ISO 8855:2011, section 7.1.8. Must be used in combination with the steeringRatio [TSteeringAngleConfiguration](#).

#### 1.13.2.4 uint64\_t TSteeringAngleData::timestamp

Timestamp of the acquisition of the steering angle signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

### 1.13.2.5 uint32\_t TSteeringAngleData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [ESteeringAngleValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [steering-angle.h](#)

## 1.14 TVehicleDataConfiguration Struct Reference

```
#include <vehicle-data.h>
```

### Public Attributes

- [EVehicleType](#) vehicleType
- [EAxleType](#) drivenAxles
- uint8\_t seatCount
- float trackWidth
- float frontAxleTrackWidth
- float wheelBase
- float vehicleMass
- float vehicleWidth
- [TDistance3D](#) distCoG2RefPoint
- [TDistance3D](#) distFrontAxle2RefPoint
- [TDistance3D](#) distRearAxle2RefPoint
- uint32\_t validityBits

### 1.14.1 Detailed Description

Static configuration data of the vehicle which are relevant for positioning.

### 1.14.2 Member Data Documentation

#### 1.14.2.1 TDistance3D TVehicleDataConfiguration::distCoG2RefPoint

Distance of the center of gravity to the reference point in 3 dimensions. Unit: [m].

#### 1.14.2.2 TDistance3D TVehicleDataConfiguration::distFrontAxle2RefPoint

Distance of front axle to the reference point in 3 dimensions. Unit: [m].

#### 1.14.2.3 TDistance3D TVehicleDataConfiguration::distRearAxle2RefPoint

Distance of rear axle to the reference point in 3 dimensions. Unit: [m].

#### 1.14.2.4 EAxleType TVehicleDataConfiguration::drivenAxles

Type of the driven axles of the vehicle.

#### 1.14.2.5 float TVehicleDataConfiguration::frontAxleTrackWidth

Front axle track width of the vehicle. Unit: [m].

#### 1.14.2.6 uint8\_t TVehicleDataConfiguration::seatCount

Number of seats of the vehicle.

#### 1.14.2.7 float TVehicleDataConfiguration::trackWidth

Track width of the vehicle. Unit: [m]. If the vehicle has different track widths at the front and rear axles, the rear axle track is referred to.

#### 1.14.2.8 uint32\_t TVehicleDataConfiguration::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EVehicleDataConfigurationValidityBits](#) values]. Must be checked before usage.

#### 1.14.2.9 float TVehicleDataConfiguration::vehicleMass

Mass of the vehicle not including the current load. Unit: [kg].

#### 1.14.2.10 EVehicleType TVehicleDataConfiguration::vehicleType

Type of the vehicle.

#### 1.14.2.11 float TVehicleDataConfiguration::vehicleWidth

Width of the vehicle. Unit: [m].

#### 1.14.2.12 float TVehicleDataConfiguration::wheelBase

Wheel base of the vehicle. Unit: [m]. The wheel base is basically the distance between the front axle and the rear axle. For an exact definition, see ISO 8855.

The documentation for this struct was generated from the following file:

- [vehicle-data.h](#)

## 1.15 TVehicleSpeedData Struct Reference

```
#include <vehicle-speed.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- float [vehicleSpeed](#)
- uint32\_t [validityBits](#)

#### 1.15.1 Detailed Description

Vehicle speed sensor service provides the current speed of the vehicle.

#### 1.15.2 Member Data Documentation

##### 1.15.2.1 uint64\_t TVehicleSpeedData::timestamp

Timestamp of the acquisition of the vehicle speed signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

##### 1.15.2.2 uint32\_t TVehicleSpeedData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EVehicleSpeedValidityBits](#) values]. Must be checked before usage.

### 1.15.2.3 float TVehicleSpeedData::vehicleSpeed

Filtered vehicle speed in [m/s] with a frequency of at least 5Hz. Direction is given by the sign of this value.

The documentation for this struct was generated from the following file:

- [vehicle-speed.h](#)

## 1.16 TVehicleStateData Struct Reference

```
#include <vehicle-state.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- bool [antiLockBrakeSystemActive](#)
- bool [brakeActive](#)
- bool [electronicStabilityProgramActive](#)
- bool [tractionControlActive](#)
- uint32\_t [validityBits](#)

### 1.16.1 Detailed Description

The VehicleStateData delivers the sensor values of the vehicle state. You must check the valid bits before usage.

### 1.16.2 Member Data Documentation

#### 1.16.2.1 bool TVehicleStateData::antiLockBrakeSystemActive

If true and signal is valid the ABS is currently engaged.

#### 1.16.2.2 bool TVehicleStateData::brakeActive

If true and signal is valid the brakes are currently engaged.

#### 1.16.2.3 bool TVehicleStateData::electronicStabilityProgramActive

If true and signal is valid the electronic stability system (ESP or DSC) is currently engaged.

#### 1.16.2.4 uint64\_t TVehicleStateData::timestamp

Timestamp of the acquisition of the accelerometer signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

#### 1.16.2.5 bool TVehicleStateData::tractionControlActive

If true and signal is valid the traction control (ASR) is currently engaged.

#### 1.16.2.6 uint32\_t TVehicleStateData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EVehicleStateValidityBits](#) values]. Must be checked before usage.

The documentation for this struct was generated from the following file:

- [vehicle-state.h](#)

## 1.17 TWheelConfiguration Struct Reference

```
#include <wheel.h>
```

### Public Attributes

- [EWheelUnit wheelUnit](#)
- [EWheelId wheel1Id](#)
- [EWheelId wheel2Id](#)
- [EWheelId wheel3Id](#)
- [EWheelId wheel4Id](#)
- [uint16\\_t wheelticksPerRevolution](#)
- [float tireRollingCircumference](#)
- [uint32\\_t validityBits](#)

#### 1.17.1 Detailed Description

Static configuration data for the wheel sensor service.

#### 1.17.2 Member Data Documentation

##### 1.17.2.1 float TWheelConfiguration::tireRollingCircumference

Distance travelled on the ground per a single revolution of one wheel. Unit: [m].

##### 1.17.2.2 uint32\_t TWheelConfiguration::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EWheelConfigValidityBits](#) values]. Must be checked before usage. Note: wheelUnit and wheel1 .. wheel4 are always valid. Therefore no dedicated validity-Bits are required.

##### 1.17.2.3 EWheelId TWheelConfiguration::wheel1Id

Identifies the wheel of the vehicle assigned to [TWheelData::wheel1](#)

##### 1.17.2.4 EWheelId TWheelConfiguration::wheel2Id

Identifies the wheel of the vehicle assigned to [TWheelData::wheel2](#)

##### 1.17.2.5 EWheelId TWheelConfiguration::wheel3Id

Identifies the wheel of the vehicle assigned to [TWheelData::wheel3](#)

##### 1.17.2.6 EWheelId TWheelConfiguration::wheel4Id

Identifies the wheel of the vehicle assigned to [TWheelData::wheel4](#)

##### 1.17.2.7 uint16\_t TWheelConfiguration::wheelticksPerRevolution

Number of ticks for a single revolution of one wheel

##### 1.17.2.8 EWheelUnit TWheelConfiguration::wheelUnit

Measurement unit in which the wheel rotation data is provided. Valid for all wheels.

The documentation for this struct was generated from the following file:

- [wheel.h](#)



## 1.18 TWheelData Struct Reference

```
#include <wheel.h>
```

### Public Attributes

- uint64\_t [timestamp](#)
- float [wheel1](#)
- float [wheel2](#)
- float [wheel3](#)
- float [wheel4](#)
- uint32\_t [statusBits](#)
- uint32\_t [validityBits](#)

### 1.18.1 Detailed Description

Wheel rotation data information for multiple wheels. Container with timestamp as used by callback and get function.

Wheel rotation data is provided for up to 4 wheels. The assignment of the fields `wheel1`, `wheel2`, `wheel3`, `wheel4` to the wheels of the vehicle and the measurement unit is provided as static configuration by the function [snsWheelGetConfiguration\(\)](#).

### 1.18.2 Member Data Documentation

#### 1.18.2.1 uint32\_t TWheelData::statusBits

Bit mask providing additional status information. [bitwise or'ed [EWheelStatusBits](#) values].

#### 1.18.2.2 uint64\_t TWheelData::timestamp

Timestamp of the acquisition of this wheel tick signal [ms]. All sensor/GNSS timestamps must be based on the same time source.

#### 1.18.2.3 uint32\_t TWheelData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EWheelValidityBits](#) values]. Must be checked before usage.

#### 1.18.2.4 float TWheelData::wheel1

Wheel rotation data for the wheel identified by `TWheelConfiguration::wheel1` in measurement unit identified by [TWheelConfiguration::wheelUnit](#).

#### 1.18.2.5 float TWheelData::wheel2

Wheel rotation data for the wheel identified by `TWheelConfiguration::wheel2` in measurement unit identified by [TWheelConfiguration::wheelUnit](#).

#### 1.18.2.6 float TWheelData::wheel3

Wheel rotation data for the wheel identified by `TWheelConfiguration::wheel3` in measurement unit identified by [TWheelConfiguration::wheelUnit](#).

#### 1.18.2.7 float TWheelData::wheel4

Wheel rotation data for the wheel identified by `TWheelConfiguration::wheel4` in measurement unit identified by [TWheelConfiguration::wheelUnit](#).

The documentation for this struct was generated from the following file:

- [wheel.h](#)

## 2 File Documentation

### 2.1 acceleration.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

#### Classes

- struct [TAccelerationConfiguration](#)
- struct [TAccelerationData](#)

#### Typedefs

- typedef void(\* [AccelerationCallback](#) )(const [TAccelerationData](#) accelerationData[], uint16\_t numElements)

#### Enumerations

- enum [EAccelerationConfigValidityBits](#) {  
[ACCELERATION\\_CONFIG\\_DISTX\\_VALID](#) = 0x00000001, [ACCELERATION\\_CONFIG\\_DISTY\\_VALID](#) = 0x00000002, [ACCELERATION\\_CONFIG\\_DISTZ\\_VALID](#) = 0x00000004, [ACCELERATION\\_CONFIG\\_ANGLEYAW\\_VALID](#) = 0x00000008,  
[ACCELERATION\\_CONFIG\\_ANGLEPITCH\\_VALID](#) = 0x00000010, [ACCELERATION\\_CONFIG\\_ANGLEROLL\\_VALID](#) = 0x00000020, [ACCELERATION\\_CONFIG\\_SIGMAX\\_VALID](#) = 0x00000040, [ACCELERATION\\_CONFIG\\_SIGMAY\\_VALID](#) = 0x00000080,  
[ACCELERATION\\_CONFIG\\_SIGMAZ\\_VALID](#) = 0x00000100, [ACCELERATION\\_CONFIG\\_TYPE\\_VALID](#) = 0x00000200 }
- enum [EAccelerationTypeBits](#) { [ACCELERATION\\_X\\_PROVIDED](#) = 0x00000001, [ACCELERATION\\_Y\\_PROVIDED](#) = 0x00000002, [ACCELERATION\\_Z\\_PROVIDED](#) = 0x00000004, [ACCELERATION\\_TEMPERATURE\\_PROVIDED](#) = 0x00000008 }
- enum [EAccelerationValidityBits](#) { [ACCELERATION\\_X\\_VALID](#) = 0x00000001, [ACCELERATION\\_Y\\_VALID](#) = 0x00000002, [ACCELERATION\\_Z\\_VALID](#) = 0x00000004, [ACCELERATION\\_TEMPERATURE\\_VALID](#) = 0x00000008 }

#### Functions

- bool [snsAccelerationInit](#) ()
- bool [snsAccelerationDestroy](#) ()
- bool [snsAccelerationGetMetaData](#) (TSensorMetaData \*data)
- bool [snsAccelerationGetAccelerationConfiguration](#) (TAccelerationConfiguration \*config)
- bool [snsAccelerationGetAccelerationData](#) (TAccelerationData \*accelerationData)
- bool [snsAccelerationRegisterCallback](#) (AccelerationCallback callback)
- bool [snsAccelerationDeregisterCallback](#) (AccelerationCallback callback)
- bool [snsAccelerationGetStatus](#) (TSensorStatus \*status)
- bool [snsAccelerationRegisterStatusCallback](#) (SensorStatusCallback callback)
- bool [snsAccelerationDeregisterStatusCallback](#) (SensorStatusCallback callback)

## 2.1.1 Typedef Documentation

### 2.1.1.1 typedef void(\* AccelerationCallback)(const TAccelerationData accelerationData[], uint16\_t numElements)

Callback type for acceleration sensor service. Use this type of callback if you want to register for acceleration data. This callback may return buffered data (`numElements > 1`) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (`numElements > 1`), the elements will be ordered with rising timestamps

#### Parameters

<i>accelerationData</i>	pointer to an array of <a href="#">TAccelerationData</a> with size <code>numElements</code>
<i>numElements</i>	allowed range: $\geq 1$ . If <code>numElements &gt; 1</code> , buffered data are provided.

## 2.1.2 Enumeration Type Documentation

### 2.1.2.1 enum EAccelerationConfigValidityBits

[TAccelerationConfiguration::validityBits](#) provides information about the currently valid signals of the acceleration configuration data. It is a or'ed bitmask of the `EAccelerationConfigValidityBits` values.

#### Enumerator

- `ACCELERATION_CONFIG_DISTX_VALID`** Validity bit for field [TAccelerationConfiguration::dist2RefPointX](#).
- `ACCELERATION_CONFIG_DISTY_VALID`** Validity bit for field [TAccelerationConfiguration::dist2RefPointY](#).
- `ACCELERATION_CONFIG_DISTZ_VALID`** Validity bit for field [TAccelerationConfiguration::dist2RefPointZ](#).
- `ACCELERATION_CONFIG_ANGLEYAW_VALID`** Validity bit for field [TAccelerationConfiguration::angleYaw](#).
- `ACCELERATION_CONFIG_ANGLEPITCH_VALID`** Validity bit for field [TAccelerationConfiguration::anglePitch](#).
- `ACCELERATION_CONFIG_ANGLEROLL_VALID`** Validity bit for field [TAccelerationConfiguration::angleRoll](#).
- `ACCELERATION_CONFIG_SIGMAX_VALID`** Validity bit for field [TAccelerationConfiguration::sigmaX](#).
- `ACCELERATION_CONFIG_SIGMAY_VALID`** Validity bit for field [TAccelerationConfiguration::sigmaX](#).
- `ACCELERATION_CONFIG_SIGMAZ_VALID`** Validity bit for field [TAccelerationConfiguration::sigmaZ](#).
- `ACCELERATION_CONFIG_TYPE_VALID`** Validity bit for field [TAccelerationConfiguration::typeBits](#).

### 2.1.2.2 enum EAccelerationTypeBits

Accelerometer type [TAccelerationConfiguration::typeBits](#) provides information about the type of the accelerometer and the interpretation of the signals. It is a or'ed bitmask of the `EAccelerationTypeBits` values.

#### Enumerator

- `ACCELERATION_X_PROVIDED`** An acceleration measurement for the x-axis is provided
- `ACCELERATION_Y_PROVIDED`** An acceleration measurement for the y-axis is provided
- `ACCELERATION_Z_PROVIDED`** An acceleration measurement for the z-axis is provided
- `ACCELERATION_TEMPERATURE_PROVIDED`** A measurement for the temperature is provided

### 2.1.2.3 enum EAccelerationValidityBits

[TAccelerationData::validityBits](#) provides information about the currently valid signals of the acceleration data. It is a or'ed bitmask of the `EAccelerationValidityBits` values. Note:

- The general availability of the signals is provided per [TAccelerationConfiguration::typeBits](#).

- If a signal is generally provided but temporarily not available, then the corresponding typeBit will be set but not the validityBit

#### Enumerator

**ACCELERATION\_X\_VALID** Validity bit for field [TAccelerationData::x](#).  
**ACCELERATION\_Y\_VALID** Validity bit for field [TAccelerationData::y](#).  
**ACCELERATION\_Z\_VALID** Validity bit for field [TAccelerationData::z](#).  
**ACCELERATION\_TEMPERATURE\_VALID** Validity bit for field [TAccelerationData::temperature](#).

### 2.1.3 Function Documentation

#### 2.1.3.1 bool snsAccelerationDeregisterCallback ( AccelerationCallback callback )

Deregister acceleration callback. After calling this method no new acceleration data will be delivered to the client.

##### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

##### Returns

True if callback has been deregistered successfully.

#### 2.1.3.2 bool snsAccelerationDeregisterStatusCallback ( SensorStatusCallback callback )

Deregister acceleration sensor status callback. After calling this method no new acceleration sensor status updates will be delivered to the client.

##### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

##### Returns

True if callback has been deregistered successfully.

#### 2.1.3.3 bool snsAccelerationDestroy ( )

Destroy the acceleration sensor service. Must be called after using the acceleration sensor service to shut down the service.

##### Returns

True if shutdown has been successful.

#### 2.1.3.4 bool snsAccelerationGetAccelerationConfiguration ( TAccelerationConfiguration \* config )

Accessing static configuration information about the acceleration sensor.

##### Parameters

<i>config</i>	After calling the method the currently available acceleration configuration data is written into this parameter.
---------------	------------------------------------------------------------------------------------------------------------------

##### Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.1.3.5 bool snsAccelerationGetAccelerationData ( TAccelerationData \* *accelerationData* )**

Method to get the acceleration data at a specific point in time. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

## Parameters

<i>accelerationData</i>	After calling the method the currently available acceleration data is written into acceleration-Data.
-------------------------	-------------------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

2.1.3.6 bool snsAccelerationGetMetaData ( **TSensorMetaData** \* *data* )

Provide meta information about sensor service. The meta data of a sensor service provides information about it's name, version, type, subtype, sampling frequency etc.

## Parameters

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

## Returns

True if meta data is available.

2.1.3.7 bool snsAccelerationGetStatus ( **TSensorStatus** \* *status* )

Method to get the acceleration sensor status at a specific point in time.

## Parameters

<i>status</i>	After calling the method the current acceleration sensor status is written into status
---------------	----------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

## 2.1.3.8 bool snsAccelerationInit ( )

Initialization of the acceleration sensor service. Must be called before using the acceleration sensor service to set up the service.

## Returns

True if initialization has been successful.

2.1.3.9 bool snsAccelerationRegisterCallback ( **AccelerationCallback** *callback* )

Register acceleration callback. This is the recommended method for continuously accessing the acceleration data. The callback will be invoked when new acceleration data is available. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

2.1.3.10 bool snsAccelerationRegisterStatusCallback ( **SensorStatusCallback** *callback* )

Register acceleration sensor status callback. This is the recommended method for continuously monitoring the acceleration sensor status. The callback will be invoked when new acceleration sensor status data is available.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

## 2.2 gyroscope.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

## Classes

- struct [TGyroscopeConfiguration](#)
- struct [TGyroscopeData](#)

## Typedefs

- typedef void(\* [GyroscopeCallback](#) )(const [TGyroscopeData](#) gyroData[], uint16\_t numElements)

## Enumerations

- enum [EGyroscopeConfigValidityBits](#) {  
[GYROSCOPE\\_CONFIG\\_ANGLEYAW\\_VALID](#) = 0x00000001, [GYROSCOPE\\_CONFIG\\_ANGLEPITCH\\_VALID](#) = 0x00000002, [GYROSCOPE\\_CONFIG\\_ANGLEROLL\\_VALID](#) = 0x00000004, [GYROSCOPE\\_CONFIG\\_MOMENTYAW\\_VALID](#) = 0x00000008,  
[GYROSCOPE\\_CONFIG\\_SIGMAGYROSCOPE\\_VALID](#) = 0x00000010, [GYROSCOPE\\_CONFIG\\_TYPE\\_VALID](#) = 0x00000020 }
- enum [EGyroscopeTypeBits](#) {  
[GYROSCOPE\\_TEMPERATURE\\_COMPENSATED](#) = 0x00000001, [GYROSCOPE\\_YAWRATE\\_PROVIDED](#) = 0x00000002, [GYROSCOPE\\_PITCHRATE\\_PROVIDED](#) = 0x00000004, [GYROSCOPE\\_ROLLRATE\\_PROVIDED](#) = 0x00000008,  
[GYROSCOPE\\_TEMPERATURE\\_PROVIDED](#) = 0x00000010 }
- enum [EGyroscopeValidityBits](#) { [GYROSCOPE\\_YAWRATE\\_VALID](#) = 0x00000001, [GYROSCOPE\\_PITCHRATE\\_VALID](#) = 0x00000002, [GYROSCOPE\\_ROLLRATE\\_VALID](#) = 0x00000004, [GYROSCOPE\\_TEMPERATURE\\_VALID](#) = 0x00000008 }

## Functions

- bool [snsGyroscopeInit](#) ()
- bool [snsGyroscopeDestroy](#) ()
- bool [snsGyroscopeGetMetaData](#) ([TSensorMetaData](#) \*data)
- bool [snsGyroscopeGetConfiguration](#) ([TGyroscopeConfiguration](#) \*gyroConfig)
- bool [snsGyroscopeGetGyroscopeData](#) ([TGyroscopeData](#) \*gyroData)
- bool [snsGyroscopeRegisterCallback](#) ([GyroscopeCallback](#) callback)
- bool [snsGyroscopeDeregisterCallback](#) ([GyroscopeCallback](#) callback)
- bool [snsGyroscopeGetStatus](#) ([TSensorStatus](#) \*status)
- bool [snsGyroscopeRegisterStatusCallback](#) ([SensorStatusCallback](#) callback)
- bool [snsGyroscopeDeregisterStatusCallback](#) ([SensorStatusCallback](#) callback)

### 2.2.1 Typedef Documentation

#### 2.2.1.1 typedef void(\* GyroscopeCallback)(const TGyroscopeData gyroData[], uint16\_t numElements)

Callback type for gyroscope sensor service. Use this type of callback if you want to register for gyroscope data. This callback may return buffered data (`numElements > 1`) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (`numElements > 1`), the elements will be ordered with rising timestamps

##### Parameters

<i>gyroData</i>	pointer to an array of <a href="#">TGyroscopeData</a> with size <code>numElements</code>
<i>numElements</i>	allowed range: $\geq 1$ . If <code>numElements &gt; 1</code> , buffered data are provided.

### 2.2.2 Enumeration Type Documentation

#### 2.2.2.1 enum EGyroscopeConfigValidityBits

[TGyroscopeConfiguration::validityBits](#) provides information about the currently valid signals of the gyroscope configuration data. It is a or'ed bitmask of the `EGyroscopeConfigValidityBits` values.

##### Enumerator

- `GYROSCOPE_CONFIG_ANGLEYAW_VALID`** Validity bit for field [TGyroscopeConfiguration::angleYaw](#).
- `GYROSCOPE_CONFIG_ANGLEPITCH_VALID`** Validity bit for field [TGyroscopeConfiguration::anglePitch](#).
- `GYROSCOPE_CONFIG_ANGLEROLL_VALID`** Validity bit for field [TGyroscopeConfiguration::angleRoll](#).
- `GYROSCOPE_CONFIG_MOMENTYAW_VALID`** Validity bit for field [TGyroscopeConfiguration::momentOfYawInertia](#).
- `GYROSCOPE_CONFIG_SIGMAGYROSCOPE_VALID`** Validity bit for field [TGyroscopeConfiguration::sigmaGyroscope](#).
- `GYROSCOPE_CONFIG_TYPE_VALID`** Validity bit for field [TGyroscopeConfiguration::typeBits](#).

#### 2.2.2.2 enum EGyroscopeTypeBits

Gyroscope type [TGyroscopeConfiguration::typeBits](#) provides information about the type of the gyroscope and the interpretation of the signals. It is a or'ed bitmask of the `EGyroscopeTypeBits` values.

##### Enumerator

- `GYROSCOPE_TEMPERATURE_COMPENSATED`** Temperature bias compensation already applied to gyroscope signals.
- `GYROSCOPE_YAWRATE_PROVIDED`** A measurement for the z/yaw-axis is provided
- `GYROSCOPE_PITCHRATE_PROVIDED`** A measurement for the y/pitch-axis is provided
- `GYROSCOPE_ROLLRATE_PROVIDED`** A measurement for the x/roll-axis is provided
- `GYROSCOPE_TEMPERATURE_PROVIDED`** A measurement for the temperature is provided

#### 2.2.2.3 enum EGyroscopeValidityBits

[TGyroscopeData::validityBits](#) provides information which fields in [TGyroscopeData](#) contain valid measurement data. It is a or'ed bitmask of the `EGyroscopeValidityBits` values. Note:

- The general availability of the signals is provided per [TGyroscopeConfiguration::typeBits](#).
- If a signal is generally provided but temporarily not available, then the corresponding typeBit will be set but not the validityBit



## Enumerator

**GYROSCOPE\_YAWRATE\_VALID** Validity bit for field [TGyroscopeData::yawRate](#).

**GYROSCOPE\_PITCHRATE\_VALID** Validity bit for field [TGyroscopeData::pitchRate](#).

**GYROSCOPE\_ROLLRATE\_VALID** Validity bit for field [TGyroscopeData::rollRate](#).

**GYROSCOPE\_TEMPERATURE\_VALID** Validity bit for field [TGyroscopeData::temperature](#).

## 2.2.3 Function Documentation

## 2.2.3.1 bool snsGyroscopeDeregisterCallback ( GyroscopeCallback callback )

Deregister gyroscope callback. After calling this method no new gyroscope data will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

## 2.2.3.2 bool snsGyroscopeDeregisterStatusCallback ( SensorStatusCallback callback )

Deregister gyroscope status callback. After calling this method no new gyroscope status updates will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

## 2.2.3.3 bool snsGyroscopeDestroy ( )

Destroy the gyroscope sensor service. Must be called after using the gyroscope sensor service to shut down the service.

## Returns

True if shutdown has been successful.

## 2.2.3.4 bool snsGyroscopeGetConfiguration ( TGyroscopeConfiguration \* gyroConfig )

Accessing static configuration information about the gyroscope sensor.

## Parameters

<i>gyroConfig</i>	After calling the method the currently available gyroscope configuration data is written into gyroConfig.
-------------------	-----------------------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

### 2.2.3.5 bool snsGyroscopeGetGyroscopeData ( TGyroscopeData \* *gyroData* )

Method to get the gyroscope data at a specific point in time. Be careful when using this method to read data often enough as gyro data are rotation rates per second. The recommended usage for the gyroscope data is the callback interface. The get method is provided for consistency reasons of the sensor service API and might be used for determining the rotation rate in certain situations. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

**Parameters**

<i>gyroData</i>	After calling the method the currently available gyroscope data is written into gyroData.
-----------------	-------------------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.2.3.6 bool snsGyroscopeGetMetaData ( TSensorMetaData \* data )**

Provide meta information about sensor service.

**Parameters**

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

**Returns**

True if meta data is available.

**2.2.3.7 bool snsGyroscopeGetStatus ( TSensorStatus \* status )**

Method to get the gyroscope status at a specific point in time.

**Parameters**

<i>status</i>	After calling the method the current gyroscope status is written into status
---------------	------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.2.3.8 bool snsGyroscopeInit ( )**

Initialization of the gyroscope sensor service. Must be called before using the gyroscope sensor service to set up the service.

**Returns**

True if initialization has been successful.

**2.2.3.9 bool snsGyroscopeRegisterCallback ( GyroscopeCallback callback )**

Register gyroscope callback. This is the recommended method for continuously accessing the gyroscope data, i.e. rotation rates. The callback will be invoked when new gyroscope data is available. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.2.3.10 bool snsGyroscopeRegisterStatusCallback ( SensorStatusCallback callback )**

Register gyroscope status callback. This is the recommended method for continuously monitoring the gyroscope status. The callback will be invoked when new gyroscope status data is available.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

## 2.3 inclination.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

## Classes

- struct [TInclinationData](#)

## Typedefs

- typedef void(\* [InclinationCallback](#) )(const [TInclinationData](#) inclinationData[], uint16\_t numElements)

## Enumerations

- enum [EInclinationSensorStatus](#) { [INCLINATION\\_INITIALIZING](#), [INCLINATION\\_SENSOR\\_BASED](#), [INCLINATION\\_UNSTABLE\\_DRIVING\\_CONDITION](#), [INCLINATION\\_MODEL\\_BASED](#), [INCLINATION\\_NOT\\_AVAILABLE](#) }
- enum [EInclinationValidityBits](#) { [INCLINATION\\_LONGITUDINAL\\_VALID](#) = 0x00000001, [INCLINATION\\_TRANSVERSE\\_VALID](#) = 0x00000002, [INCLINATION\\_STATUS\\_VALID](#) = 0x00000004 }

## Functions

- bool [snsInclinationInit](#) ()
- bool [snsInclinationDestroy](#) ()
- bool [snsInclinationGetMetaData](#) ([TSensorMetaData](#) \*data)
- bool [snsInclinationGetInclinationData](#) ([TInclinationData](#) \*inclination)
- bool [snsInclinationRegisterCallback](#) ([InclinationCallback](#) callback)
- bool [snsInclinationDeregisterCallback](#) ([InclinationCallback](#) callback)
- bool [snsInclinationGetStatus](#) ([TSensorStatus](#) \*status)
- bool [snsInclinationRegisterStatusCallback](#) ([SensorStatusCallback](#) callback)
- bool [snsInclinationDeregisterStatusCallback](#) ([SensorStatusCallback](#) callback)

## 2.3.1 Typedef Documentation

2.3.1.1 typedef void(\* [InclinationCallback](#))(const [TInclinationData](#) inclinationData[], uint16\_t numElements)

Callback type for inclination sensor service. Use this type of callback if you want to register for inclination data. This callback may return buffered data (numElements > 1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (numElements > 1), the elements will be ordered with rising timestamps

## Parameters

<i>inclinationData</i>	pointer to an array of <a href="#">TInclinationData</a> with size numElements
<i>numElements</i>	allowed range: $\geq 1$ . If numElements $> 1$ , buffered data are provided.

## 2.3.2 Enumeration Type Documentation

## 2.3.2.1 enum EInclinationSensorStatus

Current status of the inclination sensors signals.

## Enumerator

**INCLINATION\_INITIALIZING** System is currently in the initialization phase.

**INCLINATION\_SENSOR\_BASED** Signals are available and are based on sensors.

**INCLINATION\_UNSTABLE\_DRIVING\_CONDITION** Unstable driving conditions (limited accuracy).

**INCLINATION\_MODEL\_BASED** Signals are available and are based on a model.

**INCLINATION\_NOT\_AVAILABLE** Signals are not available.

## 2.3.2.2 enum EInclinationValidityBits

[TInclinationData::validityBits](#) provides information about the currently valid signals of the inclination data. It is a or'ed bitmask of the EInclinationValidityBits values.

## Enumerator

**INCLINATION\_LONGITUDINAL\_VALID** Validity bit for field [TInclinationData::longitudinalGradientRoadway](#).

**INCLINATION\_TRAVERSE\_VALID** Validity bit for field [TInclinationData::traverseGradientRoadway](#).

**INCLINATION\_STATUS\_VALID** Validity bit for field [TInclinationData::status](#).

## 2.3.3 Function Documentation

2.3.3.1 bool snsInclinationDeregisterCallback ( [InclinationCallback](#) *callback* )

Deregister inclination callback. After calling this method no new inclination data will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

2.3.3.2 bool snsInclinationDeregisterStatusCallback ( [SensorStatusCallback](#) *callback* )

Deregister inclination sensor status callback. After calling this method no new inclination sensor status updates will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

### 2.3.3.3 bool snsInclinationDestroy ( )

Destroy the inclination sensor service. Must be called after using the inclination sensor service to shut down the service.

#### Returns

True if shutdown has been successful.

### 2.3.3.4 bool snsInclinationGetInclinationData ( **TInclinationData** \* *inclination* )

Method to get the inclination data at a specific point in time. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

#### Parameters

<i>inclination</i>	After calling the method the currently available inclination data is written into this parameter.
--------------------	---------------------------------------------------------------------------------------------------

#### Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

### 2.3.3.5 bool snsInclinationGetMetaData ( **TSensorMetaData** \* *data* )

Provide meta information about sensor service. The meta data of a sensor service provides information about its name, version, type, subtype, sampling frequency etc.

#### Parameters

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

#### Returns

True if meta data is available.

### 2.3.3.6 bool snsInclinationGetStatus ( **TSensorStatus** \* *status* )

Method to get the inclination sensor status at a specific point in time.

#### Parameters

<i>status</i>	After calling the method the current inclination sensor status is written into status
---------------	---------------------------------------------------------------------------------------

#### Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

### 2.3.3.7 bool snsInclinationInit ( )

Initialization of the inclination sensor service. Must be called before using the inclination sensor service to set up the service.

#### Returns

True if initialization has been successful.

### 2.3.3.8 bool snsInclinationRegisterCallback ( **InclinationCallback** *callback* )

Register inclination callback. This is the recommended method for continuously accessing the inclination data. The callback will be invoked when new inclination data is available. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.3.3.9 bool snsInclinationRegisterStatusCallback ( [SensorStatusCallback](#) *callback* )**

Register inclination sensor status callback. This is the recommended method for continuously monitoring the inclination sensor status. The callback will be invoked when new inclination sensor status data is available.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.4 odometer.h File Reference**

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

**Classes**

- struct [TOdometerData](#)

**Typedefs**

- typedef void(\* [OdometerCallback](#) )(const [TOdometerData](#) odometerData[], uint16\_t numElements)

**Enumerations**

- enum [EOdometerValidityBits](#) { [ODOMETER\\_TRAVELLEDDISTANCE\\_VALID](#) = 0x00000001 }

**Functions**

- bool [snsOdometerInit](#) ()
- bool [snsOdometerDestroy](#) ()
- bool [snsOdometerGetMetaData](#) ([TSensorMetaData](#) \*data)
- bool [snsOdometerGetOdometerData](#) ([TOdometerData](#) \*odometer)
- bool [snsOdometerRegisterCallback](#) ([OdometerCallback](#) callback)
- bool [snsOdometerDeregisterCallback](#) ([OdometerCallback](#) callback)
- bool [snsOdometerGetStatus](#) ([TSensorStatus](#) \*status)
- bool [snsOdometerRegisterStatusCallback](#) ([SensorStatusCallback](#) callback)
- bool [snsOdometerDeregisterStatusCallback](#) ([SensorStatusCallback](#) callback)

### 2.4.1 Typedef Documentation

#### 2.4.1.1 typedef void(\* OdometerCallback)(const TOdometerData odometerData[], uint16\_t numElements)

Callback type for odometer sensor service. Use this type of callback if you want to register for odometer data. This callback may return buffered data (`numElements > 1`) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (`numElements > 1`), the elements will be ordered with rising timestamps

##### Parameters

<i>odometerData</i>	pointer to an array of <a href="#">TOdometerData</a> with size <code>numElements</code>
<i>numElements</i>	allowed range: $\geq 1$ . If <code>numElements &gt; 1</code> , buffered data are provided.

### 2.4.2 Enumeration Type Documentation

#### 2.4.2.1 enum EOdometerValidityBits

[TOdometerData::validityBits](#) provides information about the currently valid signals of the odometer data. It is a or'ed bitmask of the `EOdometerValidityBits` values.

##### Enumerator

**ODOMETER\_TRAVELLEDDISTANCE\_VALID** Validity bit for field [TOdometerData::travelledDistance](#).

### 2.4.3 Function Documentation

#### 2.4.3.1 bool snsOdometerDeregisterCallback ( OdometerCallback callback )

Deregister odometer callback. After calling this method no new odometer data will be delivered to the client.

##### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

##### Returns

True if callback has been deregistered successfully.

#### 2.4.3.2 bool snsOdometerDeregisterStatusCallback ( SensorStatusCallback callback )

Deregister odometer sensor status callback. After calling this method no new odometer sensor status updates will be delivered to the client.

##### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

##### Returns

True if callback has been deregistered successfully.

#### 2.4.3.3 bool snsOdometerDestroy ( )

Destroy the odometer sensor service. Must be called after using the odometer sensor service to shut down the service.

##### Returns

True if shutdown has been successful.



**2.4.3.4 bool snsOdometerGetMetaData ( TSensorMetaData \* data )**

Provide meta information about sensor service. The meta data of a sensor service provides information about it's name, version, type, subtype, sampling frequency etc.

## Parameters

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

## Returns

True if meta data is available.

**2.4.3.5 bool snsOdometerGetOdometerData ( TOdometerData \* odometer )**

Method to get the odometer data at a specific point in time. Be careful when using this method to read data often enough to avoid missing an overflow. With reasonable car speeds it should be enough to read the data every 3s. The recommended usage for the odometer data is the callback interface. The get method is provided for consistency reasons of the sensor service API and might be used for determining the distance between a few points in time. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

## Parameters

<i>odometer</i>	After calling the method the currently available odometer data is written into this parameter.
-----------------	------------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.4.3.6 bool snsOdometerGetStatus ( TSensorStatus \* status )**

Method to get the odometer sensor status at a specific point in time.

## Parameters

<i>status</i>	After calling the method the current odometer sensor status is written into status
---------------	------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.4.3.7 bool snsOdometerInit ( )**

Initialization of the odometer sensor service. Must be called before using the odometer sensor service to set up the service.

## Returns

True if initialization has been successful.

**2.4.3.8 bool snsOdometerRegisterCallback ( OdometerCallback callback )**

Register odometer callback. This is the recommended method for continuously accessing the odometer data. The callback will be invoked when new odometer data is available. Overflow handling must be done by the caller. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

**2.4.3.9 bool snsOdometerRegisterStatusCallback ( SensorStatusCallback callback )**

Register odometer sensor status callback. This is the recommended method for continuously monitoring the odometer sensor status. The callback will be invoked when new odometer sensor status data is available.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

## 2.5 reverse-gear.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

## Classes

- struct [TReverseGearData](#)

## Typedefs

- typedef void(\* [ReverseGearCallback](#) )(const [TReverseGearData](#) reverseGearData[], uint16\_t numElements)

## Enumerations

- enum [ERReverseGearValidityBits](#) { [REVERSEGEAR\\_REVERSEGEAR\\_VALID](#) = 0x00000001 }

## Functions

- bool [snsReverseGearInit](#) ()
- bool [snsReverseGearDestroy](#) ()
- bool [snsReverseGearGetMetaData](#) ([TSensorMetaData](#) \*data)
- bool [snsReverseGearGetReverseGearData](#) ([TReverseGearData](#) \*reverseGear)
- bool [snsReverseGearRegisterCallback](#) ([ReverseGearCallback](#) callback)
- bool [snsReverseGearDeregisterCallback](#) ([ReverseGearCallback](#) callback)
- bool [snsReverseGearGetStatus](#) ([TSensorStatus](#) \*status)
- bool [snsReverseGearRegisterStatusCallback](#) ([SensorStatusCallback](#) callback)
- bool [snsReverseGearDeregisterStatusCallback](#) ([SensorStatusCallback](#) callback)

## 2.5.1 Typedef Documentation

2.5.1.1 typedef void(\* [ReverseGearCallback](#))(const [TReverseGearData](#) reverseGearData[], uint16\_t numElements)

Callback type for reverse gear sensor service. Use this type of callback if you want to register for reverse gear data. This callback may return buffered data (numElements > 1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (numElements > 1), the elements will be ordered with rising timestamps

## Parameters

<i>reverseGear-Data</i>	pointer to an array of <a href="#">TReverseGearData</a> with size numElements
-------------------------	-------------------------------------------------------------------------------

<i>numElements</i>	allowed range: $\geq 1$ . If <i>numElements</i> $> 1$ , buffered data are provided.
--------------------	-------------------------------------------------------------------------------------

## 2.5.2 Enumeration Type Documentation

### 2.5.2.1 enum EReverseGearValidityBits

[TRReverseGearData::validityBits](#) provides information about the currently valid signals of the reverse gear data. It is a or'ed bitmask of the EReverseGearValidityBits values.

#### Enumerator

**REVERSEGEAR\_REVERSEGEAR\_VALID** Validity bit for field [TRReverseGearData::isReverseGear](#).

## 2.5.3 Function Documentation

### 2.5.3.1 bool snsReverseGearDeregisterCallback ( ReverseGearCallback callback )

Deregister reverse gear callback. After calling this method no new reverse gear data will be delivered to the client.

#### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

#### Returns

True if callback has been deregistered successfully.

### 2.5.3.2 bool snsReverseGearDeregisterStatusCallback ( SensorStatusCallback callback )

Deregister reverse gear sensor status callback. After calling this method no new reverse gear sensor status updates will be delivered to the client.

#### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

#### Returns

True if callback has been deregistered successfully.

### 2.5.3.3 bool snsReverseGearDestroy ( )

Destroy the ReverseGear sensor service. Must be called after using the ReverseGear sensor service to shut down the service.

#### Returns

True if shutdown has been successful.

### 2.5.3.4 bool snsReverseGearGetMetaData ( TSensorMetaData \* data )

Provide meta information about sensor service. The meta data of a sensor service provides information about its name, version, type, subtype, sampling frequency etc.

**Parameters**

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

**Returns**

True if meta data is available.

**2.5.3.5 bool snsReverseGearGetReverseGearData ( TReverseGearData \* *reverseGear* )**

Method to get the reverse gear data at a specific point in time. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

**Parameters**

<i>reverseGear</i>	After calling the method the currently available reverse gear data is written into reverseGear.
--------------------	-------------------------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.5.3.6 bool snsReverseGearGetStatus ( TSensorStatus \* *status* )**

Method to get the reverse gear sensor status at a specific point in time.

**Parameters**

<i>status</i>	After calling the method the current reverse gear sensor status is written into status
---------------	----------------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.5.3.7 bool snsReverseGearInit ( )**

Initialization of the reverse gear sensor service. Must be called before using the reverse gear sensor service to set up the service.

**Returns**

True if initialization has been successful.

**2.5.3.8 bool snsReverseGearRegisterCallback ( ReverseGearCallback *callback* )**

Register reverse gear callback. The callback will be invoked when new reverse gear data is available. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.5.3.9 bool snsReverseGearRegisterStatusCallback ( SensorStatusCallback *callback* )**

Register reverse gear sensor status callback. This is the recommended method for continuously monitoring the reverse gear sensor status. The callback will be invoked when new reverse gear sensor status data is available.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

## 2.6 slip-angle.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

## Classes

- struct [TSlipAngleData](#)

## Typedefs

- typedef void(\* [SlipAngleCallback](#) )(const [TSlipAngleData](#) slipAngleData[], uint16\_t numElements)

## Enumerations

- enum [ESlipAngleValidityBits](#) { [SLIPANGLE\\_SLIPANGLE\\_VALID](#) = 0x00000001 }

## Functions

- bool [snsSlipAngleInit](#) ()
- bool [snsSlipAngleDestroy](#) ()
- bool [snsSlipAngleGetMetaData](#) ([TSensorMetaData](#) \*data)
- bool [snsSlipAngleGetSlipAngleData](#) ([TSlipAngleData](#) \*data)
- bool [snsSlipAngleRegisterCallback](#) ([SlipAngleCallback](#) callback)
- bool [snsSlipAngleDeregisterCallback](#) ([SlipAngleCallback](#) callback)
- bool [snsSlipAngleGetStatus](#) ([TSensorStatus](#) \*status)
- bool [snsSlipAngleRegisterStatusCallback](#) ([SensorStatusCallback](#) callback)
- bool [snsSlipAngleDeregisterStatusCallback](#) ([SensorStatusCallback](#) callback)

## 2.6.1 Typedef Documentation

## 2.6.1.1 typedef void(\* SlipAngleCallback)(const TSlipAngleData slipAngleData[], uint16\_t numElements)

Callback type for slip angle sensor service. Use this type of callback if you want to register for slip angle data. This callback may return buffered data (numElements >1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (numElements >1), the elements will be ordered with rising timestamps

## Parameters

<i>slipAngleData</i>	pointer to an array of <a href="#">TSlipAngleData</a> with size numElements
----------------------	-----------------------------------------------------------------------------

<i>numElements</i>	allowed range: $\geq 1$ . If <i>numElements</i> $> 1$ , buffered data are provided.
--------------------	-------------------------------------------------------------------------------------

## 2.6.2 Enumeration Type Documentation

### 2.6.2.1 enum ESkipAngleValidityBits

[TSkipAngleData::validityBits](#) provides information about the currently valid signals of the skip angle data. It is a or'ed bitmask of the ESkipAngleValidityBits values.

#### Enumerator

**SLIPANGLE\_SLIPANGLE\_VALID** Validity bit for field [TSkipAngleData::skipAngle](#).

## 2.6.3 Function Documentation

### 2.6.3.1 bool snsSkipAngleDeregisterCallback ( SkipAngleCallback callback )

Deregister skip angle callback. After calling this method no new skip angle data will be delivered to the client.

#### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

#### Returns

True if callback has been deregistered successfully.

### 2.6.3.2 bool snsSkipAngleDeregisterStatusCallback ( SensorStatusCallback callback )

Deregister skip angle sensor status callback. After calling this method no new skip angle sensor status updates will be delivered to the client.

#### Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

#### Returns

True if callback has been deregistered successfully.

### 2.6.3.3 bool snsSkipAngleDestroy ( )

Destroy the SkipAngle sensor service. Must be called after using the SkipAngle sensor service to shut down the service.

#### Returns

True if shutdown has been successful.

### 2.6.3.4 bool snsSkipAngleGetMetaData ( TSensorMetaData \* data )

Provide meta information about sensor service. The meta data of a sensor service provides information about its name, version, type, subtype, sampling frequency etc.

## Parameters

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

## Returns

True if meta data is available.

**2.6.3.5 bool snsSlipAngleGetSlipAngleData ( TSlipAngleData \* *data* )**

Method to get the slip angle data at a specific point in time. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

## Parameters

<i>data</i>	After calling the method the currently available inclination data is written into this parameter.
-------------	---------------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.6.3.6 bool snsSlipAngleGetStatus ( TSensorStatus \* *status* )**

Method to get the slip angle sensor status at a specific point in time.

## Parameters

<i>status</i>	After calling the method the current slip angle sensor status is written into status
---------------	--------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.6.3.7 bool snsSlipAngleInit ( )**

Initialization of the slip angle sensor service. Must be called before using the slip angle sensor service to set up the service.

## Returns

True if initialization has been successful.

**2.6.3.8 bool snsSlipAngleRegisterCallback ( SlipAngleCallback *callback* )**

Register slip angle callback. This is the recommended method for continuously accessing the slip angle data. The callback will be invoked when new slip angle data is available. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

**2.6.3.9 bool snsSlipAngleRegisterStatusCallback ( SensorStatusCallback *callback* )**

Register slip angle sensor status callback. This is the recommended method for continuously monitoring the slip angle sensor status. The callback will be invoked when new slip angle sensor status data is available.



**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.7 sns-init.h File Reference**

```
#include <stdint.h>
#include <stdbool.h>
```

**Macros**

- `#define GENIVI_SNS_API_MAJOR 4`
- `#define GENIVI_SNS_API_MINOR 0`
- `#define GENIVI_SNS_API_MICRO 0`

**Functions**

- `bool snsInit ()`
- `bool snsDestroy ()`
- `void snsGetVersion (int *major, int *minor, int *micro)`

**2.7.1 Macro Definition Documentation****2.7.1.1 #define GENIVI\_SNS\_API\_MAJOR 4****2.7.1.2 #define GENIVI\_SNS\_API\_MICRO 0****2.7.1.3 #define GENIVI\_SNS\_API\_MINOR 0****2.7.2 Function Documentation****2.7.2.1 bool snsDestroy ( )**

Destroy the sensor services. Must be called after using the sensor services for shut down. After this call no sensor will be able to receive sensor data. The individual sensors must be shut down before this call. Otherwise system behaviour is not defined.

**Returns**

True if shutdown has been successful.

**2.7.2.2 void snsGetVersion ( int \* major, int \* minor, int \* micro )**

Sensor services version information. This information is for the sensor services system structure. The version information for each sensor can be obtained via the metadata.

**Parameters**

<i>major</i>	Major version number. Changes in this number are used for incompatible API change.
<i>minor</i>	Minor version number. Changes in this number are used for compatible API change.
<i>micro</i>	Micro version number. Changes in this number are used for minor changes.

### 2.7.2.3 bool snsinit ( )

Initialization of the sensor service system infrastructure. Must be called before using any of the sensor services to set up general structures and connections to the sensors or signal providers. If not called before a sensor init the system behaviour is not defined.

#### Returns

True if initialization has been successful.

## 2.8 sns-meta-data.h File Reference

```
#include <stdint.h>
```

#### Classes

- struct [TSensorMetaData](#)

#### Enumerations

- enum [ESensorCategory](#) { [SENSOR\\_CATEGORY\\_UNKNOWN](#), [SENSOR\\_CATEGORY\\_LOGICAL](#), [SENSOR\\_CATEGORY\\_PHYSICAL](#) }
- enum [ESensorType](#) { [SENSOR\\_TYPE\\_UNKNOWN](#), [SENSOR\\_TYPE\\_ACCELERATION](#), [SENSOR\\_TYPE\\_GYROSCOPE](#), [SENSOR\\_TYPE\\_INCLINATION](#), [SENSOR\\_TYPE\\_ODOMETER](#), [SENSOR\\_TYPE\\_REVERSE\\_GEAR](#), [SENSOR\\_TYPE\\_SLIP\\_ANGLE](#), [SENSOR\\_TYPE\\_STEERING\\_ANGLE](#), [SENSOR\\_TYPE\\_VEHICLE\\_SPEED](#), [SENSOR\\_TYPE\\_VEHICLE\\_STATE](#), [SENSOR\\_TYPE\\_WHEELTICK](#), [SENSOR\\_TYPE\\_WHEELSPEEDANGULAR](#), [SENSOR\\_TYPE\\_WHEELSPEED](#) }

#### Functions

- int32\_t [getSensorMetaDataList](#) (const [TSensorMetaData](#) \*\*metadata)

### 2.8.1 Enumeration Type Documentation

#### 2.8.1.1 enum ESensorCategory

The sensor category introduces the concept that sensor information can also be derived information computed by combining several signals.

#### Enumerator

**[SENSOR\\_CATEGORY\\_UNKNOWN](#)** Unknown category. Should not be used.

**[SENSOR\\_CATEGORY\\_LOGICAL](#)** A logical sensor can combine signals of several sensors.

**[SENSOR\\_CATEGORY\\_PHYSICAL](#)** A physical sensor provides signals from physically available sensors.

### 2.8.1.2 enum ESensorType

The sensor type identifies which physical quantity is measured. For each sensor type, there is a corresponding API header with data structures, callback notifications and getter functions defined. Note that for all 3 wheel sensor types there is a combined API header.

#### Enumerator

**SENSOR\_TYPE\_UNKNOWN** Unknown sensor type. Should not be used.  
**SENSOR\_TYPE\_ACCELERATION** Acceleration sensor  
**SENSOR\_TYPE\_GYROSCOPE** Gyroscope sensor  
**SENSOR\_TYPE\_INCLINATION** Inclination sensor  
**SENSOR\_TYPE\_ODOMETER** Odometer sensor  
**SENSOR\_TYPE\_REVERSE\_GEAR** Reverse gear sensor  
**SENSOR\_TYPE\_SLIP\_ANGLE** Slip angle sensor  
**SENSOR\_TYPE\_STEERING\_ANGLE** Steering angle sensor  
**SENSOR\_TYPE\_VEHICLE\_SPEED** Vehicle speed sensor  
**SENSOR\_TYPE\_VEHICLE\_STATE** Vehicle state sensor  
**SENSOR\_TYPE\_WHELTICK** Wheel tick sensor  
**SENSOR\_TYPE\_WHEELSPEEDANGULAR** Wheel speed angular sensor  
**SENSOR\_TYPE\_WHEELSPEED** Wheel speed sensor

### 2.8.2 Function Documentation

#### 2.8.2.1 int32\_t getSensorMetaDataList ( const TSensorMetaData \*\* metadata )

Retrieve the metadata of all available sensors.

#### Parameters

<i>metadata</i>	returns a pointer an array of <a href="#">TSensorMetaData</a> (maybe NULL if no metadata is available)
-----------------	--------------------------------------------------------------------------------------------------------

#### Returns

number of elements in the array of [TSensorMetaData](#)

## 2.9 sns-status.h File Reference

#### Classes

- struct [TSensorStatus](#)

#### Typedefs

- typedef void(\* [SensorStatusCallback](#) )(const [TSensorStatus](#) \*status)

#### Enumerations

- enum [ESensorStatus](#) {  
[SENSOR\\_STATUS\\_NOTAVAILABLE](#) = 0, [SENSOR\\_STATUS\\_INITIALIZING](#) = 1, [SENSOR\\_STATUS\\_AVAILABLE](#) = 2, [SENSOR\\_STATUS\\_RESTARTING](#) = 3,  
[SENSOR\\_STATUS\\_FAILURE](#) = 4, [SENSOR\\_STATUS\\_OUTOFSERVICE](#) = 5 }
- enum [ESensorStatusValidityBits](#) { [SENSOR\\_STATUS\\_STATUS\\_VALID](#) = 0x00000001 }

### 2.9.1 Typedef Documentation

#### 2.9.1.1 `typedef void(* SensorStatusCallback)(const TSensorStatus *status)`

Callback type for sensor status. Use this type of callback if you want to register for sensor status updates data.

## Parameters

<i>status</i>	the sensor status
---------------	-------------------

## 2.9.2 Enumeration Type Documentation

## 2.9.2.1 enum ESensorStatus

Enumeration to describe the status of the sensor

## Enumerator

**SENSOR\_STATUS\_NOTAVAILABLE** Sensor is not available at all, based on configuration data.

**SENSOR\_STATUS\_INITIALIZING** Initial status when the connection to the Sensor is set up for the first time.

**SENSOR\_STATUS\_AVAILABLE** Sensor is available and running as expected.

**SENSOR\_STATUS\_RESTARTING** Sensor is restarted, i.e. due to communication loss.

**SENSOR\_STATUS\_FAILURE** Sensor is not operating properly. Restarting did not help.

**SENSOR\_STATUS\_OUTOFSERVICE** Sensor is temporarily not available, due to some known external condition, vehicle bus or external ECU providing the signal being off.

## 2.9.2.2 enum ESensorStatusValidityBits

[TSensorStatus::validityBits](#) provides information about the currently valid signals of the [TSensorStatus](#) struct. It is a or'ed bitmask of the ESensorStatusValidityBits values.

## Enumerator

**SENSOR\_STATUS\_STATUS\_VALID** Validity bit for field [TSensorStatus::status](#).

## 2.10 steering-angle.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

## Classes

- struct [TSteeringAngleData](#)
- struct [TSteeringAngleConfiguration](#)

## Typedefs

- typedef void(\* [SteeringAngleCallback](#) )(const [TSteeringAngleData](#) steeringAngleData[], uint16\_t numElements)

## Enumerations

- enum [ESteeringAngleValidityBits](#) { [STEERINGANGLE\\_FRONT\\_VALID](#) = 0x00000001, [STEERINGANGLE\\_REAR\\_VALID](#) = 0x00000002, [STEERINGANGLE\\_STEERINGWHEEL\\_VALID](#) = 0x00000004 }

## Functions

- bool [snsSteeringAngleInit](#) ()
- bool [snsSteeringAngleDestroy](#) ()
- bool [snsSteeringAngleGetMetaData](#) (TSensorMetaData \*data)
- bool [snsSteeringAngleGetSteeringAngleData](#) (TSteeringAngleData \*angleData)
- bool [snsSteeringAngleGetConfiguration](#) (TSteeringAngleConfiguration \*config)
- bool [snsSteeringAngleRegisterCallback](#) (SteeringAngleCallback callback)
- bool [snsSteeringAngleDeregisterCallback](#) (SteeringAngleCallback callback)
- bool [snsSteeringAngleGetStatus](#) (TSensorStatus \*status)
- bool [snsSteeringAngleRegisterStatusCallback](#) (SensorStatusCallback callback)
- bool [snsSteeringAngleDeregisterStatusCallback](#) (SensorStatusCallback callback)

## 2.10.1 Typedef Documentation

## 2.10.1.1 typedef void(\* SteeringAngleCallback)(const TSteeringAngleData steeringAngleData[], uint16\_t numElements)

Callback type for steering angle sensor service. Use this type of callback if you want to register for steering angle data. This callback may return buffered data (numElements >1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (numElements >1), the elements will be ordered with rising timestamps

## Parameters

<i>steeringAngle-Data</i>	pointer to an array of <a href="#">TSteeringAngleData</a> with size numElements
<i>numElements</i>	allowed range: >=1. If numElements >1, buffered data are provided.

## 2.10.2 Enumeration Type Documentation

## 2.10.2.1 enum ESteeringAngleValidityBits

[TSteeringAngleData::validityBits](#) provides information about the currently valid signals of the steering angle data. It is a or'ed bitmask of the ESteeringAngleValidityBits values.

## Enumerator

**STEERINGANGLE\_FRONT\_VALID** Validity bit for field [TSteeringAngleData::front](#).

**STEERINGANGLE\_REAR\_VALID** Validity bit for field [TSteeringAngleData::rear](#).

**STEERINGANGLE\_STEERINGWHEEL\_VALID** Validity bit for field [TSteeringAngleData::steeringWheel](#).

## 2.10.3 Function Documentation

## 2.10.3.1 bool snsSteeringAngleDeregisterCallback ( SteeringAngleCallback callback )

Deregister steering angle callback. After calling this method no new steering angle data will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

#### 2.10.3.2 `bool snsSteeringAngleDeregisterStatusCallback ( SensorStatusCallback callback )`

Deregister steering angle sensor status callback. After calling this method no new steering angle sensor status updates will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

## 2.10.3.3 bool snsSteeringAngleDestroy ( )

Destroy the SteeringAngle sensor service. Must be called after using the SteeringAngle sensor service to shut down the service.

## Returns

True if shutdown has been successful.

## 2.10.3.4 bool snsSteeringAngleGetConfiguration ( TSteeringAngleConfiguration \* config )

Accessing static configuration information about the steering angle sensor.

## Parameters

<i>config</i>	After calling the method the currently available static steering angle configuration data is written into config.
---------------	-------------------------------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

## 2.10.3.5 bool snsSteeringAngleGetMetaData ( TSensorMetaData \* data )

Provide meta information about sensor service. The meta data of a sensor service provides information about it's name, version, type, subtype, sampling frequency etc.

## Parameters

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

## Returns

True if meta data is available.

## 2.10.3.6 bool snsSteeringAngleGetStatus ( TSensorStatus \* status )

Method to get the steering angle sensor status at a specific point in time.

## Parameters

<i>status</i>	After calling the method the current steering angle sensor status is written into status
---------------	------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

## 2.10.3.7 bool snsSteeringAngleGetSteeringAngleData ( TSteeringAngleData \* angleData )

Method to get the steering angle data at a specific point in time. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.



**Parameters**

<i>angleData</i>	After calling the method the currently available steering angle data is written into angleData.
------------------	-------------------------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.10.3.8 bool snsSteeringAngleInit ( )**

Initialization of the steering angle sensor service. Must be called before using the steering angle sensor service to set up the service.

**Returns**

True if initialization has been successful.

**2.10.3.9 bool snsSteeringAngleRegisterCallback ( SteeringAngleCallback callback )**

Register steering angle callback. This is the recommended method for continuously accessing the steering angle data. The callback will be invoked when new steering angle data is available. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.10.3.10 bool snsSteeringAngleRegisterStatusCallback ( SensorStatusCallback callback )**

Register steering angle sensor status callback. This is the recommended method for continuously monitoring the steering angle sensor status. The callback will be invoked when new steering angle sensor status data is available.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.11 vehicle-data.h File Reference**

```
#include <stdint.h>
#include <stdbool.h>
```

**Classes**

- struct [TDistance3D](#)
- struct [TVehicleDataConfiguration](#)

## Enumerations

- enum `EAxleType` { `SNS_AXLE_UNKNOWN` = 0, `SNS_AXLE_FRONT` = 1, `SNS_AXLE_REAR` = 2, `SNS_AXLE_BOTH` = 3 }
- enum `EVehicleType` { `SNS_CAR` = 0, `SNS_TRUCK` = 1, `SNS_MOTORBIKE` = 2, `SNS_BUS` = 3 }
- enum `EVehicleDataConfigurationValidityBits` {  
`VEHICLESTATUS_VEHICLETYPE_VALID` = 0x00000001, `VEHICLESTATUS_DRIVENAXLES_VALID` = 0x00000002, `VEHICLESTATUS_SEATCOUNT_VALID` = 0x00000004, `VEHICLESTATUS_TRACKWIDTH_VALID` = 0x00000008,  
`VEHICLESTATUS_FRONTAXLETRACKWIDTH_VALID` = 0x00000010, `VEHICLESTATUS_WHELLBASE_VALID` = 0x00000020, `VEHICLESTATUS_VEHICLEMASS_VALID` = 0x00000040, `VEHICLESTATUS_VEHICLEWIDTH_VALID` = 0x00000080,  
`VEHICLESTATUS_DISTCOG2REFPOINT_VALID` = 0x00000100, `VEHICLESTATUS_DISTFRONTAXLE2REFPOINT_VALID` = 0x00000200, `VEHICLESTATUS_DISTREARAXLE2REFPOINT_VALID` = 0x00000400  
}

## Functions

- bool `snsVehicleDataInit` ()
- bool `snsVehicleDataDestroy` ()
- bool `snsVehicleDataGetConfiguration` (`TVehicleDataConfiguration` \*vehicleDataConfiguration)

## 2.11.1 Enumeration Type Documentation

2.11.1.1 enum `EAxleType`

Description of driven axles. This is currently restricted to passenger cars.

## Enumerator

**`SNS_AXLE_UNKNOWN`** It is not known which axles are driven.

**`SNS_AXLE_FRONT`** Only the front axle is driven.

**`SNS_AXLE_REAR`** Only the rear axle is driven.

**`SNS_AXLE_BOTH`** Both axles are driven (4 wheel drive).

2.11.1.2 enum `EVehicleDataConfigurationValidityBits`

`TVehicleData::validityBits` provides information about the valid fields of the vehicle data. It is a or'ed bitmask of the `EVehicleDataConfigurationValidityBits` values.

## Enumerator

**`VEHICLESTATUS_VEHICLETYPE_VALID`** Validity bit for field `TVehicleDataConfiguration::vehicleType`.

**`VEHICLESTATUS_DRIVENAXLES_VALID`** Validity bit for field `TVehicleDataConfiguration::drivenAxles`.

**`VEHICLESTATUS_SEATCOUNT_VALID`** Validity bit for field `TVehicleDataConfiguration::seatCount`.

**`VEHICLESTATUS_TRACKWIDTH_VALID`** Validity bit for field `TVehicleDataConfiguration::trackWidth`.

**`VEHICLESTATUS_FRONTAXLETRACKWIDTH_VALID`** Validity bit for field `TVehicleDataConfiguration::frontAxleTrackWidth`.

**`VEHICLESTATUS_WHELLBASE_VALID`** Validity bit for field `TVehicleDataConfiguration::wheelBase`.

**`VEHICLESTATUS_VEHICLEMASS_VALID`** Validity bit for field `TVehicleDataConfiguration::vehicleMass`.

**`VEHICLESTATUS_VEHICLEWIDTH_VALID`** Validity bit for field `TVehicleDataConfiguration::vehicleWidth`.

**`VEHICLESTATUS_DISTCOG2REFPOINT_VALID`** Validity bit for field `TVehicleDataConfiguration::distCoG2RefPoint`.

**VEHICLESTATUS\_DISTFRONTAXLE2REFPOINT\_VALID** Validity bit for field [TVehicleDataConfiguration::distFrontAxle2RefPoint](#).

**VEHICLESTATUS\_DISTREARAXLE2REFPOINT\_VALID** Validity bit for field [TVehicleDataConfiguration::vehicleType](#).

### 2.11.1.3 enum EVehicleType

Description of the vehicle type. This is for future extensions. Currently the specification is based mostly on cars. Other vehicle types are just referenced for future extensions.

#### Enumerator

**SNS\_CAR** Passenger car with 4 wheels.

**SNS\_TRUCK** Truck

**SNS\_MOTORBIKE** Motorbike with 2 wheels.

**SNS\_BUS** Passenger bus.

### 2.11.2 Function Documentation

#### 2.11.2.1 bool snsVehicleDataDestroy ( )

Destroy the vehicle data sensor service. Must be called after using the vehicle data sensor service to shut down the service.

#### Returns

True if shutdown has been successful.

#### 2.11.2.2 bool snsVehicleDataGetConfiguration ( TVehicleDataConfiguration \* vehicleDataConfiguration )

Accessing static configuration information about the vehicle.

#### Parameters

<i>vehicleData</i>	After calling the method the available vehicle configuration data is written into vehicleData.
--------------------	------------------------------------------------------------------------------------------------

#### Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

#### 2.11.2.3 bool snsVehicleDataInit ( )

Initialization of the vehicle data sensor service. Must be called before using the vehicle data sensor service to set up the service.

#### Returns

True if initialization has been successful.

## 2.12 vehicle-speed.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

## Classes

- struct [TVehicleSpeedData](#)

## Typedefs

- typedef void(\* [VehicleSpeedCallback](#) )(const [TVehicleSpeedData](#) vehicleSpeedData[], uint16\_t numElements)

## Enumerations

- enum [EVehicleSpeedValidityBits](#) { [VEHICLESPEED\\_VEHICLESPEED\\_VALID](#) = 0x00000001 }

## Functions

- bool [snsVehicleSpeedInit](#) ()
- bool [snsVehicleSpeedDestroy](#) ()
- bool [snsVehicleSpeedGetMetaData](#) ([TSensorMetaData](#) \*data)
- bool [snsVehicleSpeedGetVehicleSpeedData](#) ([TVehicleSpeedData](#) \*vehicleSpeed)
- bool [snsVehicleSpeedRegisterCallback](#) ([VehicleSpeedCallback](#) callback)
- bool [snsVehicleSpeedDeregisterCallback](#) ([VehicleSpeedCallback](#) callback)
- bool [snsVehicleSpeedGetStatus](#) ([TSensorStatus](#) \*status)
- bool [snsVehicleSpeedRegisterStatusCallback](#) ([SensorStatusCallback](#) callback)
- bool [snsVehicleSpeedDeregisterStatusCallback](#) ([SensorStatusCallback](#) callback)

## 2.12.1 Typedef Documentation

2.12.1.1 typedef void(\* [VehicleSpeedCallback](#))(const [TVehicleSpeedData](#) vehicleSpeedData[], uint16\_t numElements)

Callback type for vehicle speed sensor service. Use this type of callback if you want to register for vehicle speed data. This callback may return buffered data (numElements >1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (numElements >1), the elements will be ordered with rising timestamps

## Parameters

<i>vehicleSpeed-Data</i>	pointer to an array of <a href="#">TVehicleSpeedData</a> with size numElements
<i>numElements</i>	allowed range: >=1. If numElements >1, buffered data are provided.

## 2.12.2 Enumeration Type Documentation

2.12.2.1 enum [EVehicleSpeedValidityBits](#)

[TVehicleSpeedData::validityBits](#) provides information about the currently valid signals of the vehicle speed data. It is a or'ed bitmask of the [EVehicleSpeedValidityBits](#) values.

## Enumerator

**[VEHICLESPEED\\_VEHICLESPEED\\_VALID](#)** Validity bit for field [TVehicleSpeedData::vehicleSpeed](#).

## 2.12.3 Function Documentation

2.12.3.1 bool [snsVehicleSpeedDeregisterCallback](#) ( [VehicleSpeedCallback](#) callback )

Deregister vehicle speed callback. After calling this method no new vehicle speed data will be delivered to the client.

**Parameters**

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

**Returns**

True if callback has been deregistered successfully.

**2.12.3.2 bool snsVehicleSpeedDeregisterStatusCallback ( SensorStatusCallback callback )**

Deregister vehicle speed sensor status callback. After calling this method no new vehicle speed sensor status updates will be delivered to the client.

**Parameters**

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

**Returns**

True if callback has been deregistered successfully.

**2.12.3.3 bool snsVehicleSpeedDestroy ( )**

Destroy the VehicleSpeed sensor service. Must be called after using the VehicleSpeed sensor service to shut down the service.

**Returns**

True if shutdown has been successful.

**2.12.3.4 bool snsVehicleSpeedGetMetaData ( TSensorMetaData \* data )**

Provide meta information about sensor service. The meta data of a sensor service provides information about it's name, version, type, subtype, sampling frequency etc.

**Parameters**

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

**Returns**

True if meta data is available.

**2.12.3.5 bool snsVehicleSpeedGetStatus ( TSensorStatus \* status )**

Method to get the vehicle speed sensor status at a specific point in time.

**Parameters**

<i>status</i>	After calling the method the current vehicle speed sensor status is written into status
---------------	-----------------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.12.3.6 bool snsVehicleSpeedGetVehicleSpeedData ( TVehicleSpeedData \* vehicleSpeed )**

Method to get the vehicle speed data at a specific point in time.

## Parameters

<i>vehicleSpeed</i>	After calling the method the currently available vehicle speed data is written into vehicle-Speed.
---------------------	----------------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

## 2.12.3.7 bool snsVehicleSpeedInit ( )

Initialization of the vehicle speed sensor service. Must be called before using the vehicle speed sensor service to set up the service.

## Returns

True if initialization has been successful.

## 2.12.3.8 bool snsVehicleSpeedRegisterCallback ( VehicleSpeedCallback callback )

Register vehicle speed callback. The callback will be invoked when new vehicle speed data is available.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

## 2.12.3.9 bool snsVehicleSpeedRegisterStatusCallback ( SensorStatusCallback callback )

Register vehicle speed sensor status callback. This is the recommended method for continuously monitoring the vehicle speed sensor status. The callback will be invoked when new vehicle speed sensor status data is available.

## Parameters

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

## Returns

True if callback has been registered successfully.

## 2.13 vehicle-state.h File Reference

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

## Classes

- struct [TVehicleStateData](#)

## Typedefs

- typedef void(\* [VehicleStateCallback](#) )(const [TVehicleStateData](#) vehicleState[], uint16\_t numElements)

## Enumerations

- enum [EVehicleStateValidityBits](#) { [VEHICLESTATE\\_ANTILOCKBRAKESYSTEMACTIVE\\_VALID](#) = 0x00000001, [VEHICLESTATE\\_BRAKEACTIVE\\_VALID](#) = 0x00000002, [VEHICLESTATE\\_ELECTRONICSTABILITYPROGRAMACTIVE\\_VALID](#) = 0x00000004, [VEHICLESTATE\\_TRACTIONCONTROLACTIVE\\_VALID](#) = 0x00000008 }

## Functions

- bool [snsVehicleStateInit](#) ()
- bool [snsVehicleStateDestroy](#) ()
- bool [snsVehicleStateGetMetaData](#) (TSensorMetaData \*data)
- bool [snsVehicleStateGetVehicleStateData](#) (TVehicleStateData \*vehicleState)
- bool [snsVehicleStateRegisterCallback](#) (VehicleStateCallback callback)
- bool [snsVehicleStateDeregisterCallback](#) (VehicleStateCallback callback)
- bool [snsVehicleStateGetStatus](#) (TSensorStatus \*status)
- bool [snsVehicleStateRegisterStatusCallback](#) (SensorStatusCallback callback)
- bool [snsVehicleStateDeregisterStatusCallback](#) (SensorStatusCallback callback)

### 2.13.1 Typedef Documentation

#### 2.13.1.1 typedef void(\* VehicleStateCallback)(const TVehicleStateData vehicleState[], uint16\_t numElements)

Callback type for vehicle state sensor service. Use this type of callback if you want to register for vehicle state data. This callback may return buffered data (`numElements > 1`) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (`numElements > 1`), the elements will be ordered with rising timestamps

#### Parameters

<i>vehicleSpeed-Data</i>	pointer to an array of <a href="#">TVehicleStateData</a> with size <code>numElements</code>
<i>numElements</i>	allowed range: $\geq 1$ . If <code>numElements &gt; 1</code> , buffered data are provided.

### 2.13.2 Enumeration Type Documentation

#### 2.13.2.1 enum EVehicleStateValidityBits

[TVehicleStateData::validityBits](#) provides information about the currently valid signals of the vehicle state data. It is a or'ed bitmask of the [EVehicleStateValidityBits](#) values.

#### Enumerator

**[VEHICLESTATE\\_ANTILOCKBRAKESYSTEMACTIVE\\_VALID](#)** Validity bit for field `antiLockBrakeSystemActive::antiLockBrakeSystemActive`.

**[VEHICLESTATE\\_BRAKEACTIVE\\_VALID](#)** Validity bit for field `antiLockBrakeSystemActive::brakeActive`.

**[VEHICLESTATE\\_ELECTRONICSTABILITYPROGRAMACTIVE\\_VALID](#)** Validity bit for field `antiLockBrakeSystemActive::electronicStabilityProgramActive`.

**[VEHICLESTATE\\_TRACTIONCONTROLACTIVE\\_VALID](#)** Validity bit for field `antiLockBrakeSystemActive::tractionControlActive`.

### 2.13.3 Function Documentation

#### 2.13.3.1 bool snsVehicleStateDeregisterCallback ( VehicleStateCallback callback )

Deregister vehicle state callback. After calling this method no new vehicle state data will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

**2.13.3.2 bool snsVehicleStateDeregisterStatusCallback ( *SensorStatusCallback callback* )**

Deregister vehicle state sensor status callback. After calling this method no new vehicle state sensor status updates will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

**2.13.3.3 bool snsVehicleStateDestroy ( )**

Destroy the VehicleState sensor service. Must be called after using the VehicleState sensor service to shut down the service.

## Returns

True if shutdown has been successful.

**2.13.3.4 bool snsVehicleStateGetMetaData ( *TSensorMetaData \* data* )**

Provide meta information about sensor service. The meta data of a sensor service provides information about it's name, version, type, subtype, sampling frequency etc.

## Parameters

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

## Returns

True if meta data is available.

**2.13.3.5 bool snsVehicleStateGetStatus ( *TSensorStatus \* status* )**

Method to get the vehicle state sensor status at a specific point in time.

## Parameters

<i>status</i>	After calling the method the current vehicle state sensor status is written into status
---------------	-----------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.13.3.6 bool snsVehicleStateGetVehicleStateData ( *TVehicleStateData \* vehicleState* )**

Method to get the vehicle state data at a specific point in time. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.



**Parameters**

<i>vehicleState</i>	After calling the method the currently available vehicle state data is written into vehicleState.
---------------------	---------------------------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.13.3.7 bool snsVehicleStateInit ( )**

Initialization of the vehicle state sensor service. Must be called before using the vehicle state sensor service to set up the service.

**Returns**

True if initialization has been successful.

**2.13.3.8 bool snsVehicleStateRegisterCallback ( VehicleStateCallback callback )**

Register vehicle state callback. This is the recommended method for continuously accessing the vehicle state data. The callback will be invoked when new vehicle state data is available. All valid flags are updated. The data is only guaranteed to be updated when the valid flags are true.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.13.3.9 bool snsVehicleStateRegisterStatusCallback ( SensorStatusCallback callback )**

Register vehicle state sensor status callback. This is the recommended method for continuously monitoring the vehicle state sensor status. The callback will be invoked when new vehicle state sensor status data is available.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.14 wheel.h File Reference**

```
#include "sns-meta-data.h"
#include "sns-status.h"
#include <stdbool.h>
```

**Classes**

- struct [TWheelConfiguration](#)
- struct [TWheelData](#)

**Typedefs**

- typedef void(\* [WheelCallback](#) )(const [TWheelData](#) wheelData[], uint16\_t numElements)
- typedef void(\* [WheelStatusCallback](#) )(const [TSensorStatus](#) \*status)

## Enumerations

- enum `EWheelId` { `WHEEL_INVALID` = 0, `WHEEL_UNKNOWN` = 1, `WHEEL_AXLE_NONDRIVEN` = 2, `WHEEL_AXLE_FRONT` = 3, `WHEEL_AXLE_REAR` = 4, `WHEEL_LEFT_FRONT` = 5, `WHEEL_RIGHT_FRONT` = 6, `WHEEL_LEFT_REAR` = 7, `WHEEL_RIGHT_REAR` = 8 }
- enum `EWheelStatusBits` { `WHEEL_STATUS_GAP` = 0x00000001, `WHEEL_STATUS_INIT` = 0x00000002 }
- enum `EWheelValidityBits` { `WHEEL1_VALID` = 0x00000001, `WHEEL2_VALID` = 0x00000002, `WHEEL3_VALID` = 0x00000004, `WHEEL4_VALID` = 0x00000008 }
- enum `EWheelUnit` { `WHEEL_UNIT_TICKS` = 1, `WHEEL_UNIT_SPEED` = 2, `WHEEL_UNIT_ANGULAR_SPEED` = 3 }
- enum `EWheelConfigValidityBits` { `WHEEL_CONFIG_TICKS_PER_REV` = 0x00000001, `WHEEL_CONFIG_TIRE_CIRC` = 0x00000002 }

## Functions

- bool `snsWheelInit` ()
- bool `snsWheelDestroy` ()
- bool `snsWheelGetMetaData` (`TSensorMetaData` \*data)
- bool `snsWheelGetConfiguration` (`TWheelConfiguration` \*config)
- bool `snsWheelGetWheelData` (`TWheelData` \*wheelData)
- bool `snsWheelRegisterCallback` (`WheelCallback` callback)
- bool `snsWheelDeregisterCallback` (`WheelCallback` callback)
- bool `snsWheelGetStatus` (`TSensorStatus` \*status)
- bool `snsWheelRegisterStatusCallback` (`SensorStatusCallback` callback)
- bool `snsWheelDeregisterStatusCallback` (`SensorStatusCallback` callback)

## 2.14.1 Typedef Documentation

2.14.1.1 typedef void(\* WheelCallback)(const `TWheelData` wheelData[], uint16\_t numElements)

Callback type for wheel sensor service. Use this type of callback if you want to register for wheel rotation data. This callback may return buffered data (`numElements` > 1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (`numElements` > 1), the elements will be ordered with rising timestamps All wheel data belonging to the same timestamp will be provided in the same structure, i.e. there will be never two callbacks or array elements with the same timestamp.

## Parameters

<i>wheelData</i>	pointer to an array of <code>TWheelData</code> with size <code>numElements</code>
<i>numElements</i>	allowed range: >=1. If <code>numElements</code> > 1, buffered data are provided.

2.14.1.2 typedef void(\* WheelStatusCallback)(const `TSensorStatus` \*status)

Callback type for wheel sensor status. Use this type of callback if you want to register for wheel sensor status updates data.

## Parameters

<i>status</i>	the wheel sensor status
---------------	-------------------------

## 2.14.2 Enumeration Type Documentation

### 2.14.2.1 enum `EWheelConfigValidityBits`

`TWheelConfiguration::validityBits` provides information about the currently valid signals of the wheel configuration data. It is a or'ed bitmask of the `EWheelConfigValidityBits` values.

#### Enumerator

**`WHEEL_CONFIG_TICKS_PER_REV`** Validity bit for field `TWheelConfiguration::wheelTicksPerRevolution`.

**`WHEEL_CONFIG_TIRE_CIRC`** Validity bit for field `TWheelConfiguration::tireRollingCircumference`.

### 2.14.2.2 enum `EWheelId`

This header file defines the interface for wheel rotation data. Wheel rotation data may be provided as wheel ticks, as wheel speed or as angular wheel speed.

It is vehicle specific which kind of wheel rotation data is available and for which wheels the data is provided. The vehicle specific configuration can be queried using the `snsWheelGetConfiguration()` function.

Note: The driving direction (forward/backward) shall always be coded as sign of the wheel rotation data. This will reduce synchronization effort with separate reverse gear info by the client application which is in particular useful with buffered data. How the driving direction can be derived is vehicle specific. Defines the wheel or axle for which the wheel rotation information is provided. Usually wheel rotation data are either provided for the non-driven axle or for each wheel individually.

#### Enumerator

**`WHEEL_INVALID`** Wheel data is invalid / the field is unused.

**`WHEEL_UNKNOWN`** No information available where the wheel rotation data is from.

**`WHEEL_AXLE_NONDRIVEN`** Wheel rotation data is an average from the non driven axle. Can be front or rear depending on the type of drivetrain.

**`WHEEL_AXLE_FRONT`** Wheel rotation data is an average from the front axle.

**`WHEEL_AXLE_REAR`** Wheel rotation data is an average from the rear axle.

**`WHEEL_LEFT_FRONT`** Wheel rotation data is from the left front wheel.

**`WHEEL_RIGHT_FRONT`** Wheel rotation data is from the right front wheel.

**`WHEEL_LEFT_REAR`** Wheel rotation data is from the left rear wheel.

**`WHEEL_RIGHT_REAR`** Wheel rotation data is from the right rear wheel.

### 2.14.2.3 enum `EWheelStatusBits`

Additional status information for wheel data, in particular when provided as wheel ticks. This may help the client application to estimate the reliability of the wheel data. `TWheelData::statusBits` is a or'ed bitmask of the `EWheelStatusBits` values.

#### Background information

Wheel ticks are typically provided by the ABS/ESC ECU as rolling counters on the vehicle bus. To calculate the wheel ticks per time interval as provided by this API in the `TWheelData` structure, the difference between the two *valid* rolling counter values at end and start of the time interval has to be calculated (taking into account potential rollover). If any of the rolling counter values is invalid or if there has been a reset of the rolling counter in the time interval, then no valid difference can be calculated. Therefore an appropriate error/exception handling has to be implemented in the library translating the rolling counters from the vehicle bus to the wheel ticks per time interval provided by this API.

Besides to the validity indication provided with each wheel rotation update, the client (typically the Enhanced-PositionService) using the wheel rotation API may be interested to know whether wheel rotation information may have been lost. In such a case it could adapt its error estimation related to wheel ticks or even make an internal reset of its corresponding states. The status bits in enum `EWheelStatusBits` are defined to provide such information.

### Further Background

This section gives an additional overview about the possible signal path of wheel tick data and the resulting possible exceptional situations: There may be a gateway between the ABS/ESC ECU and the IVI system to separate vehicle networks. This gateway may reduce the update rate of the CAN messages, e.g. from 20ms cycle time sent by the ABS ECU to 100ms provided for the IVI system. When the update rate is reduced, the rolling counters may have to be resampled e.g. from 8 bit to 13 bit to avoid rollover within the update period. The rolling counters typically start from 0 when either the ABS ECU or the gateway is started/restarted, e.g. at an ignition cycle. The rolling counters are often accompanied with additional status information to indicate validity, reset conditions or to allow to detect loss of wheel tick data during transmission on vehicle bus (such as sequence numbers). This status information has to be evaluated to determine whether the difference calculation between subsequent rolling counters yields a valid result or not. The kind of status information provided alongside with the wheel ticks is very OEM specific

- sometimes even additional context information such as ignition status has to be considered. Nearly all above mentioned parameters are OEM or vehicle specific: update rate, size of rolling counters, status indications, lifecycle of ECU, gateway, vehicle bus, ... The status bits in enum `EWheelStatusBits` attempt to provide an appropriate abstraction for the relevant vehicle specific status information.

### Enumerator

**WHEEL\_STATUS\_GAP** There has been a gap in data collection, i.e. an unknown number of wheel revolutions has been lost. The reason for such a gap can be for example

- wheel tick data on the vehicle bus explicitly tagged as invalid
- interrupted reception of vehicle bus messages

**WHEEL\_STATUS\_INIT** This is the first wheel data of a bus or ignition lifecycle, i.e. the wheel tick difference calculation may be less reliable.

#### 2.14.2.4 enum EWheelUnit

Defines the measurement unit in which the wheel rotation data is provided.

The wheel rotation direction is given by the sign of the wheel rotation value: Positive values indicate forward driving. Negative values indicate backward driving.

### Enumerator

**WHEEL\_UNIT\_TICKS** Wheel rotation data is provided as number of wheel ticks accumulated within measurement interval. Note 1: Therefore, if the wheel ticks on the vehicle bus are represented as rolling counters, this is the difference between two subsequent rolling counter values taking the vehicle specific roll-over boundary into account. Note 2: It is safe to store integer values such as for wheel ticks without precision loss in float variables for values up to  $2^{23}$ .

**WHEEL\_UNIT\_SPEED** Wheel rotation data is provided as speed in [m/s].

**WHEEL\_UNIT\_ANGULAR\_SPEED** Wheel rotation data is provided as angular speed in [1/s] rotation per seconds.

#### 2.14.2.5 enum EWheelValidityBits

`TWheelData::validityBits` provides information which fields in `TWheelData` contain valid measurement data. It is a or'ed bitmask of the `EWheelValidityBits` values. Note: The assignment of the fields to the wheels of the vehicle is provided by the function `snsWheelGetConfiguration()`.

### Enumerator

**WHEEL1\_VALID** Validity bit for field `TWheelData::wheel1`.

**WHEEL2\_VALID** Validity bit for field `TWheelData::wheel2`.

**WHEEL3\_VALID** Validity bit for field `TWheelData::wheel3`.

**WHEEL4\_VALID** Validity bit for field `TWheelData::wheel4`.

### 2.14.3 Function Documentation

#### 2.14.3.1 `bool snsWheelDeregisterCallback ( WheelCallback callback )`

Deregister multiple wheel rotation data callback. After calling this method no new rotation data data will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

2.14.3.2 bool snsWheelDeregisterStatusCallback ( **SensorStatusCallback** *callback* )

Deregister wheel sensor status callback. After calling this method no new wheel sensor status updates will be delivered to the client.

## Parameters

<i>callback</i>	The callback which should be deregistered.
-----------------	--------------------------------------------

## Returns

True if callback has been deregistered successfully.

## 2.14.3.3 bool snsWheelDestroy ( )

Destroy the wheel sensor service. Must be called after using the wheel sensor service to shut down the service.

## Returns

True if shutdown has been successful.

2.14.3.4 bool snsWheelGetConfiguration ( **TWheelConfiguration** \* *config* )

Accessing static configuration information about the wheel sensor.

## Parameters

<i>config</i>	After calling the method the currently available acceleration wheel data is written into this parameter.
---------------	----------------------------------------------------------------------------------------------------------

## Returns

Is true if data can be provided and false otherwise, e.g. missing initialization

2.14.3.5 bool snsWheelGetMetaData ( **TSensorMetaData** \* *data* )

Provide meta information about sensor service. The meta data of a sensor service provides information about it's name, version, type, subtype, sampling frequency etc.

## Parameters

<i>data</i>	Meta data content about the sensor service.
-------------	---------------------------------------------

## Returns

True if meta data is available.

2.14.3.6 bool snsWheelGetStatus ( **TSensorStatus** \* *status* )

Method to get the wheel sensor status at a specific point in time.

**Parameters**

<i>status</i>	After calling the method the current wheel sensor status is written into status
---------------	---------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.14.3.7 bool snsWheelGetWheelData ( TWheelData \* wheelData )**

Method to get the wheel rotation data at a specific point in time.

**Parameters**

<i>wheelData</i>	After calling the method the currently available wheel rotation data is written into the array.
------------------	-------------------------------------------------------------------------------------------------

**Returns**

Is true if data can be provided and false otherwise, e.g. missing initialization

**2.14.3.8 bool snsWheelInit ( )**

Initialization of the wheel sensor service. Must be called before using the wheel sensor service to set up the service.

**Returns**

True if initialization has been successful.

**2.14.3.9 bool snsWheelRegisterCallback ( WheelCallback callback )**

Register callback for multiple wheel rotation data information. This is the recommended method for continuously accessing the wheel data. The callback will be invoked when new rotation data data is available.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

**2.14.3.10 bool snsWheelRegisterStatusCallback ( SensorStatusCallback callback )**

Register wheel sensor status callback. This is the recommended method for continuously monitoring the wheel sensor status. The callback will be invoked when new wheel sensor status data is available.

**Parameters**

<i>callback</i>	The callback which should be registered.
-----------------	------------------------------------------

**Returns**

True if callback has been registered successfully.

## Index

- ACCELERATION\_CONFIG\_ANGLEPITCH\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_ANGLEROLL\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_ANGLEYAW\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_DISTX\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_DISTY\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_DISTZ\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_SIGMAX\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_SIGMAY\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_SIGMAZ\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_CONFIG\_TYPE\_VALID
  - [acceleration.h, 20](#)
- ACCELERATION\_TEMPERATURE\_PROVIDED
  - [acceleration.h, 20](#)
- ACCELERATION\_TEMPERATURE\_VALID
  - [acceleration.h, 21](#)
- ACCELERATION\_X\_PROVIDED
  - [acceleration.h, 20](#)
- ACCELERATION\_X\_VALID
  - [acceleration.h, 21](#)
- ACCELERATION\_Y\_PROVIDED
  - [acceleration.h, 20](#)
- ACCELERATION\_Y\_VALID
  - [acceleration.h, 21](#)
- ACCELERATION\_Z\_PROVIDED
  - [acceleration.h, 20](#)
- ACCELERATION\_Z\_VALID
  - [acceleration.h, 21](#)
- [acceleration.h](#)
  - ACCELERATION\_CONFIG\_ANGLEPITCH\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_ANGLEROLL\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_ANGLEYAW\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_DISTX\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_DISTY\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_DISTZ\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_SIGMAX\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_SIGMAY\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_SIGMAZ\_VALID, [20](#)
  - ACCELERATION\_CONFIG\_TYPE\_VALID, [20](#)
  - ACCELERATION\_TEMPERATURE\_PROVIDED, [20](#)
  - ACCELERATION\_TEMPERATURE\_VALID, [21](#)
  - ACCELERATION\_X\_PROVIDED, [20](#)
  - ACCELERATION\_X\_VALID, [21](#)
  - ACCELERATION\_Y\_PROVIDED, [20](#)
  - ACCELERATION\_Y\_VALID, [21](#)
  - ACCELERATION\_Z\_PROVIDED, [20](#)
  - ACCELERATION\_Z\_VALID, [21](#)
- [AccelerationCallback](#)
  - [acceleration.h, 20](#)
- [anglePitch](#)
  - [TAccelerationConfiguration, 3](#)
  - [TGyroscopeConfiguration, 6](#)
- [angleRoll](#)
  - [TAccelerationConfiguration, 3](#)
  - [TGyroscopeConfiguration, 6](#)
- [angleYaw](#)
  - [TAccelerationConfiguration, 3](#)
  - [TGyroscopeConfiguration, 6](#)
- [antiLockBrakeSystemActive](#)
  - [TVehicleStateData, 16](#)
- [brakeActive](#)
  - [TVehicleStateData, 16](#)
- [category](#)
  - [TSensorMetaData, 11](#)
- [cycleTime](#)
  - [TSensorMetaData, 11](#)
- [dist2RefPointX](#)
  - [TAccelerationConfiguration, 3](#)
- [dist2RefPointY](#)
  - [TAccelerationConfiguration, 3](#)
- [dist2RefPointZ](#)
  - [TAccelerationConfiguration, 3](#)
- [distCoG2RefPoint](#)
  - [TVehicleDataConfiguration, 14](#)
- [distFrontAxle2RefPoint](#)
  - [TVehicleDataConfiguration, 14](#)
- [distRearAxle2RefPoint](#)
  - [TVehicleDataConfiguration, 14](#)
- [drivenAxles](#)
  - [TVehicleDataConfiguration, 14](#)



- EAccelerationConfigValidityBits
  - acceleration.h, [20](#)
- EAccelerationTypeBits
  - acceleration.h, [20](#)
- EAccelerationValidityBits
  - acceleration.h, [20](#)
- EAxleType
  - vehicle-data.h, [51](#)
- EGyroscopeConfigValidityBits
  - gyroscope.h, [25](#)
- EGyroscopeTypeBits
  - gyroscope.h, [25](#)
- EGyroscopeValidityBits
  - gyroscope.h, [25](#)
- EInclinationSensorStatus
  - inclination.h, [30](#)
- EInclinationValidityBits
  - inclination.h, [30](#)
- EOdometerValidityBits
  - odometer.h, [33](#)
- EReverseGearValidityBits
  - reverse-gear.h, [37](#)
- ESensorCategory
  - sns-meta-data.h, [43](#)
- ESensorStatus
  - sns-status.h, [46](#)
- ESensorStatusValidityBits
  - sns-status.h, [46](#)
- ESensorType
  - sns-meta-data.h, [43](#)
- ESlipAngleValidityBits
  - slip-angle.h, [40](#)
- ESteeringAngleValidityBits
  - steering-angle.h, [47](#)
- EVehicleDataConfigurationValidityBits
  - vehicle-data.h, [51](#)
- EVehicleSpeedValidityBits
  - vehicle-speed.h, [53](#)
- EVehicleStateValidityBits
  - vehicle-state.h, [56](#)
- EVehicleType
  - vehicle-data.h, [52](#)
- EWheelConfigValidityBits
  - wheel.h, [60](#)
- EWheelId
  - wheel.h, [60](#)
- EWheelStatusBits
  - wheel.h, [60](#)
- EWheelUnit
  - wheel.h, [61](#)
- EWheelValidityBits
  - wheel.h, [61](#)
- electronicStabilityProgramActive
  - TVehicleStateData, [16](#)
- front
  - TSteeringAngleData, [13](#)
- frontAxleTrackWidth
  - TVehicleDataConfiguration, [14](#)
- GYROSCOPE\_CONFIG\_ANGLEPITCH\_VALID
  - gyroscope.h, [25](#)
- GYROSCOPE\_CONFIG\_ANGLEROLL\_VALID
  - gyroscope.h, [25](#)
- GYROSCOPE\_CONFIG\_ANGLEYAW\_VALID
  - gyroscope.h, [25](#)
- GYROSCOPE\_CONFIG\_MOMENTYAW\_VALID
  - gyroscope.h, [25](#)
- GYROSCOPE\_CONFIG\_SIGMAGYROSCOPE\_VALID
  - gyroscope.h, [25](#)
- GYROSCOPE\_CONFIG\_TYPE\_VALID
  - gyroscope.h, [25](#)
- GYROSCOPE\_PITCHRATE\_PROVIDED
  - gyroscope.h, [25](#)
- GYROSCOPE\_PITCHRATE\_VALID
  - gyroscope.h, [26](#)
- GYROSCOPE\_ROLLRATE\_PROVIDED
  - gyroscope.h, [25](#)
- GYROSCOPE\_ROLLRATE\_VALID
  - gyroscope.h, [26](#)
- GYROSCOPE\_TEMPERATURE\_COMPENSATED
  - gyroscope.h, [25](#)
- GYROSCOPE\_TEMPERATURE\_PROVIDED
  - gyroscope.h, [25](#)
- GYROSCOPE\_TEMPERATURE\_VALID
  - gyroscope.h, [26](#)
- GYROSCOPE\_YAWRATE\_PROVIDED
  - gyroscope.h, [25](#)
- GYROSCOPE\_YAWRATE\_VALID
  - gyroscope.h, [26](#)
- getSensorMetaDataList
  - sns-meta-data.h, [44](#)
- gyroscope.h
  - GYROSCOPE\_CONFIG\_ANGLEPITCH\_VALID, [25](#)
  - GYROSCOPE\_CONFIG\_ANGLEROLL\_VALID, [25](#)
  - GYROSCOPE\_CONFIG\_ANGLEYAW\_VALID, [25](#)
  - GYROSCOPE\_CONFIG\_MOMENTYAW\_VALID, [25](#)
  - GYROSCOPE\_CONFIG\_SIGMAGYROSCOPE\_VALID, [25](#)
  - GYROSCOPE\_CONFIG\_TYPE\_VALID, [25](#)
  - GYROSCOPE\_PITCHRATE\_PROVIDED, [25](#)
  - GYROSCOPE\_PITCHRATE\_VALID, [26](#)
  - GYROSCOPE\_ROLLRATE\_PROVIDED, [25](#)
  - GYROSCOPE\_ROLLRATE\_VALID, [26](#)
  - GYROSCOPE\_TEMPERATURE\_COMPENSATED, [25](#)
  - GYROSCOPE\_TEMPERATURE\_PROVIDED, [25](#)
  - GYROSCOPE\_TEMPERATURE\_VALID, [26](#)
  - GYROSCOPE\_YAWRATE\_PROVIDED, [25](#)
  - GYROSCOPE\_YAWRATE\_VALID, [26](#)
- gyroscope.h, [24](#)
  - EGyroscopeConfigValidityBits, [25](#)
  - EGyroscopeTypeBits, [25](#)
  - EGyroscopeValidityBits, [25](#)
  - GyroscopeCallback, [25](#)
  - snsGyroscopeDeregisterCallback, [26](#)

- snsGyroscopeDeregisterStatusCallback, 26
- snsGyroscopeDestroy, 26
- snsGyroscopeGetConfiguration, 26
- snsGyroscopeGetGyroscopeData, 26
- snsGyroscopeGetMetaData, 28
- snsGyroscopeGetStatus, 28
- snsGyroscopeInit, 28
- snsGyroscopeRegisterCallback, 28
- snsGyroscopeRegisterStatusCallback, 28
- GyroscopeCallback
  - gyroscope.h, 25
- INCLINATION\_INITIALIZING
  - inclination.h, 30
- INCLINATION\_LONGITUDINAL\_VALID
  - inclination.h, 30
- INCLINATION\_MODEL\_BASED
  - inclination.h, 30
- INCLINATION\_NOT\_AVAILABLE
  - inclination.h, 30
- INCLINATION\_SENSOR\_BASED
  - inclination.h, 30
- INCLINATION\_STATUS\_VALID
  - inclination.h, 30
- INCLINATION\_TRAVERSE\_VALID
  - inclination.h, 30
- INCLINATION\_UNSTABLE\_DRIVING\_CONDITION
  - inclination.h, 30
- inclination.h
  - INCLINATION\_INITIALIZING, 30
  - INCLINATION\_LONGITUDINAL\_VALID, 30
  - INCLINATION\_MODEL\_BASED, 30
  - INCLINATION\_NOT\_AVAILABLE, 30
  - INCLINATION\_SENSOR\_BASED, 30
  - INCLINATION\_STATUS\_VALID, 30
  - INCLINATION\_TRAVERSE\_VALID, 30
  - INCLINATION\_UNSTABLE\_DRIVING\_CONDITION, 30
- inclination.h, 29
  - EInclinationSensorStatus, 30
  - EInclinationValidityBits, 30
  - InclinationCallback, 29
  - snsInclinationDeregisterCallback, 30
  - snsInclinationDeregisterStatusCallback, 30
  - snsInclinationDestroy, 30
  - snsInclinationGetInclinationData, 31
  - snsInclinationGetMetaData, 31
  - snsInclinationGetStatus, 31
  - snsInclinationInit, 31
  - snsInclinationRegisterCallback, 31
  - snsInclinationRegisterStatusCallback, 32
- InclinationCallback
  - inclination.h, 29
- isReverseGear
  - TReverseGearData, 10
- longitudinalGradientRoadway
  - TInclinationData, 8
- momentOfYawInertia
  - TGyroscopeConfiguration, 6
- ODOMETER\_TRAVELLEDDISTANCE\_VALID
  - odometer.h, 33
- odometer.h
  - ODOMETER\_TRAVELLEDDISTANCE\_VALID, 33
- odometer.h, 32
  - EOdometerValidityBits, 33
  - OdometerCallback, 33
  - snsOdometerDeregisterCallback, 33
  - snsOdometerDeregisterStatusCallback, 33
  - snsOdometerDestroy, 33
  - snsOdometerGetMetaData, 33
  - snsOdometerGetOdometerData, 35
  - snsOdometerGetStatus, 35
  - snsOdometerInit, 35
  - snsOdometerRegisterCallback, 35
  - snsOdometerRegisterStatusCallback, 35
- OdometerCallback
  - odometer.h, 33
- pitchRate
  - TGyroscopeData, 7
- REVERSEGEAR\_REVERSEGEAR\_VALID
  - reverse-gear.h, 37
- rear
  - TSteeringAngleData, 13
- reverse-gear.h
  - REVERSEGEAR\_REVERSEGEAR\_VALID, 37
- reverse-gear.h, 36
  - EReverseGearValidityBits, 37
  - ReverseGearCallback, 36
  - snsReverseGearDeregisterCallback, 37
  - snsReverseGearDeregisterStatusCallback, 37
  - snsReverseGearDestroy, 37
  - snsReverseGearGetMetaData, 37
  - snsReverseGearGetReverseGearData, 38
  - snsReverseGearGetStatus, 38
  - snsReverseGearInit, 38
  - snsReverseGearRegisterCallback, 38
  - snsReverseGearRegisterStatusCallback, 38
- ReverseGearCallback
  - reverse-gear.h, 36
- rollRate
  - TGyroscopeData, 7
- SENSOR\_CATEGORY\_LOGICAL
  - sns-meta-data.h, 43
- SENSOR\_CATEGORY\_PHYSICAL
  - sns-meta-data.h, 43
- SENSOR\_CATEGORY\_UNKNOWN
  - sns-meta-data.h, 43
- SENSOR\_STATUS\_AVAILABLE
  - sns-status.h, 46
- SENSOR\_STATUS\_FAILURE
  - sns-status.h, 46
- SENSOR\_STATUS\_INITIALIZING

- sns-status.h, 46
- SENSOR\_STATUS\_NOTAVAILABLE
  - sns-status.h, 46
- SENSOR\_STATUS\_OUTOFSERVICE
  - sns-status.h, 46
- SENSOR\_STATUS\_RESTARTING
  - sns-status.h, 46
- SENSOR\_STATUS\_STATUS\_VALID
  - sns-status.h, 46
- SENSOR\_TYPE\_ACCELERATION
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_GYROSCOPE
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_INCLINATION
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_ODOMETER
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_REVERSE\_GEAR
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_SLIP\_ANGLE
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_STEERING\_ANGLE
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_UNKNOWN
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_VEHICLE\_SPEED
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_VEHICLE\_STATE
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_WHEELSPEED
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_WHEELSPEEDANGULAR
  - sns-meta-data.h, 44
- SENSOR\_TYPE\_WHEELTICK
  - sns-meta-data.h, 44
- SLIPANGLE\_SLIPANGLE\_VALID
  - slip-angle.h, 40
- SNS\_AXLE\_BOTH
  - vehicle-data.h, 51
- SNS\_AXLE\_FRONT
  - vehicle-data.h, 51
- SNS\_AXLE\_REAR
  - vehicle-data.h, 51
- SNS\_AXLE\_UNKNOWN
  - vehicle-data.h, 51
- SNS\_BUS
  - vehicle-data.h, 52
- SNS\_CAR
  - vehicle-data.h, 52
- SNS\_MOTORBIKE
  - vehicle-data.h, 52
- SNS\_TRUCK
  - vehicle-data.h, 52
- STEERINGANGLE\_FRONT\_VALID
  - steering-angle.h, 47
- STEERINGANGLE\_REAR\_VALID
  - steering-angle.h, 47
- STEERINGANGLE\_STEERINGWHEEL\_VALID
  - steering-angle.h, 47
- seatCount
  - TVehicleDataConfiguration, 14
- SensorStatusCallback
  - sns-status.h, 45
- sigmaGyroscope
  - TGyroscopeConfiguration, 7
- sigmaSteeringAngle
  - TSteeringAngleConfiguration, 13
- sigmaSteeringWheelAngle
  - TSteeringAngleConfiguration, 13
- sigmaX
  - TAccelerationConfiguration, 3
- sigmaY
  - TAccelerationConfiguration, 3
- sigmaZ
  - TAccelerationConfiguration, 3
- slip-angle.h
  - SLIPANGLE\_SLIPANGLE\_VALID, 40
- slip-angle.h, 39
  - ESlipAngleValidityBits, 40
  - SlipAngleCallback, 39
  - snsSlipAngleDeregisterCallback, 40
  - snsSlipAngleDeregisterStatusCallback, 40
  - snsSlipAngleDestroy, 40
  - snsSlipAngleGetMetaData, 40
  - snsSlipAngleGetSlipAngleData, 41
  - snsSlipAngleGetStatus, 41
  - snsSlipAngleInit, 41
  - snsSlipAngleRegisterCallback, 41
  - snsSlipAngleRegisterStatusCallback, 41
- slipAngle
  - TSlipAngleData, 12
- SlipAngleCallback
  - slip-angle.h, 39
- sns-init.h, 42
  - snsDestroy, 42
  - snsGetVersion, 42
  - snsInit, 43
- sns-meta-data.h
  - SENSOR\_CATEGORY\_LOGICAL, 43
  - SENSOR\_CATEGORY\_PHYSICAL, 43
  - SENSOR\_CATEGORY\_UNKNOWN, 43
  - SENSOR\_TYPE\_ACCELERATION, 44
  - SENSOR\_TYPE\_GYROSCOPE, 44
  - SENSOR\_TYPE\_INCLINATION, 44
  - SENSOR\_TYPE\_ODOMETER, 44
  - SENSOR\_TYPE\_REVERSE\_GEAR, 44
  - SENSOR\_TYPE\_SLIP\_ANGLE, 44
  - SENSOR\_TYPE\_STEERING\_ANGLE, 44
  - SENSOR\_TYPE\_UNKNOWN, 44
  - SENSOR\_TYPE\_VEHICLE\_SPEED, 44
  - SENSOR\_TYPE\_VEHICLE\_STATE, 44
  - SENSOR\_TYPE\_WHEELSPEED, 44
  - SENSOR\_TYPE\_WHEELSPEEDANGULAR, 44
  - SENSOR\_TYPE\_WHEELTICK, 44
- sns-meta-data.h, 43
  - ESensorCategory, 43

- ESensorType, [43](#)
- getSensorMeta dataList, [44](#)
- sns-status.h
  - SENSOR\_STATUS\_AVAILABLE, [46](#)
  - SENSOR\_STATUS\_FAILURE, [46](#)
  - SENSOR\_STATUS\_INITIALIZING, [46](#)
  - SENSOR\_STATUS\_NOTAVAILABLE, [46](#)
  - SENSOR\_STATUS\_OUTOFSERVICE, [46](#)
  - SENSOR\_STATUS\_RESTARTING, [46](#)
  - SENSOR\_STATUS\_STATUS\_VALID, [46](#)
- sns-status.h, [44](#)
  - ESensorStatus, [46](#)
  - ESensorStatusValidityBits, [46](#)
  - SensorStatusCallback, [45](#)
- snsAccelerationDeregisterCallback
  - acceleration.h, [21](#)
- snsAccelerationDeregisterStatusCallback
  - acceleration.h, [21](#)
- snsAccelerationDestroy
  - acceleration.h, [21](#)
- snsAccelerationGetAccelerationConfiguration
  - acceleration.h, [21](#)
- snsAccelerationGetAccelerationData
  - acceleration.h, [21](#)
- snsAccelerationGetMetaData
  - acceleration.h, [23](#)
- snsAccelerationGetStatus
  - acceleration.h, [23](#)
- snsAccelerationInit
  - acceleration.h, [23](#)
- snsAccelerationRegisterCallback
  - acceleration.h, [23](#)
- snsAccelerationRegisterStatusCallback
  - acceleration.h, [23](#)
- snsDestroy
  - sns-init.h, [42](#)
- snsGetVersion
  - sns-init.h, [42](#)
- snsGyroscopeDeregisterCallback
  - gyroscope.h, [26](#)
- snsGyroscopeDeregisterStatusCallback
  - gyroscope.h, [26](#)
- snsGyroscopeDestroy
  - gyroscope.h, [26](#)
- snsGyroscopeGetConfiguration
  - gyroscope.h, [26](#)
- snsGyroscopeGetGyroscopeData
  - gyroscope.h, [26](#)
- snsGyroscopeGetMetaData
  - gyroscope.h, [28](#)
- snsGyroscopeGetStatus
  - gyroscope.h, [28](#)
- snsGyroscopeInit
  - gyroscope.h, [28](#)
- snsGyroscopeRegisterCallback
  - gyroscope.h, [28](#)
- snsGyroscopeRegisterStatusCallback
  - gyroscope.h, [28](#)
- snsInclinationDeregisterCallback
  - inclination.h, [30](#)
- snsInclinationDeregisterStatusCallback
  - inclination.h, [30](#)
- snsInclinationDestroy
  - inclination.h, [30](#)
- snsInclinationGetInclinationData
  - inclination.h, [31](#)
- snsInclinationGetMetaData
  - inclination.h, [31](#)
- snsInclinationGetStatus
  - inclination.h, [31](#)
- snsInclinationInit
  - inclination.h, [31](#)
- snsInclinationRegisterCallback
  - inclination.h, [31](#)
- snsInclinationRegisterStatusCallback
  - inclination.h, [32](#)
- snsInit
  - sns-init.h, [43](#)
- snsOdometerDeregisterCallback
  - odometer.h, [33](#)
- snsOdometerDeregisterStatusCallback
  - odometer.h, [33](#)
- snsOdometerDestroy
  - odometer.h, [33](#)
- snsOdometerGetMetaData
  - odometer.h, [33](#)
- snsOdometerGetOdometerData
  - odometer.h, [35](#)
- snsOdometerGetStatus
  - odometer.h, [35](#)
- snsOdometerInit
  - odometer.h, [35](#)
- snsOdometerRegisterCallback
  - odometer.h, [35](#)
- snsOdometerRegisterStatusCallback
  - odometer.h, [35](#)
- snsReverseGearDeregisterCallback
  - reverse-gear.h, [37](#)
- snsReverseGearDeregisterStatusCallback
  - reverse-gear.h, [37](#)
- snsReverseGearDestroy
  - reverse-gear.h, [37](#)
- snsReverseGearGetMetaData
  - reverse-gear.h, [37](#)
- snsReverseGearGetReverseGearData
  - reverse-gear.h, [38](#)
- snsReverseGearGetStatus
  - reverse-gear.h, [38](#)
- snsReverseGearInit
  - reverse-gear.h, [38](#)
- snsReverseGearRegisterCallback
  - reverse-gear.h, [38](#)
- snsReverseGearRegisterStatusCallback
  - reverse-gear.h, [38](#)
- snsSlipAngleDeregisterCallback
  - slip-angle.h, [40](#)

- snsSlipAngleDeregisterStatusCallback
  - slip-angle.h, [40](#)
- snsSlipAngleDestroy
  - slip-angle.h, [40](#)
- snsSlipAngleGetMetaData
  - slip-angle.h, [40](#)
- snsSlipAngleGetSlipAngleData
  - slip-angle.h, [41](#)
- snsSlipAngleGetStatus
  - slip-angle.h, [41](#)
- snsSlipAngleInit
  - slip-angle.h, [41](#)
- snsSlipAngleRegisterCallback
  - slip-angle.h, [41](#)
- snsSlipAngleRegisterStatusCallback
  - slip-angle.h, [41](#)
- snsSteeringAngleDeregisterCallback
  - steering-angle.h, [47](#)
- snsSteeringAngleDeregisterStatusCallback
  - steering-angle.h, [47](#)
- snsSteeringAngleDestroy
  - steering-angle.h, [49](#)
- snsSteeringAngleGetConfiguration
  - steering-angle.h, [49](#)
- snsSteeringAngleGetMetaData
  - steering-angle.h, [49](#)
- snsSteeringAngleGetStatus
  - steering-angle.h, [49](#)
- snsSteeringAngleGetSteeringAngleData
  - steering-angle.h, [49](#)
- snsSteeringAngleInit
  - steering-angle.h, [50](#)
- snsSteeringAngleRegisterCallback
  - steering-angle.h, [50](#)
- snsSteeringAngleRegisterStatusCallback
  - steering-angle.h, [50](#)
- snsVehicleDataDestroy
  - vehicle-data.h, [52](#)
- snsVehicleDataGetConfiguration
  - vehicle-data.h, [52](#)
- snsVehicleDataInit
  - vehicle-data.h, [52](#)
- snsVehicleSpeedDeregisterCallback
  - vehicle-speed.h, [53](#)
- snsVehicleSpeedDeregisterStatusCallback
  - vehicle-speed.h, [54](#)
- snsVehicleSpeedDestroy
  - vehicle-speed.h, [54](#)
- snsVehicleSpeedGetMetaData
  - vehicle-speed.h, [54](#)
- snsVehicleSpeedGetStatus
  - vehicle-speed.h, [54](#)
- snsVehicleSpeedGetVehicleSpeedData
  - vehicle-speed.h, [54](#)
- snsVehicleSpeedInit
  - vehicle-speed.h, [55](#)
- snsVehicleSpeedRegisterCallback
  - vehicle-speed.h, [55](#)
- snsVehicleSpeedRegisterStatusCallback
  - vehicle-speed.h, [55](#)
- snsVehicleStateDeregisterCallback
  - vehicle-state.h, [56](#)
- snsVehicleStateDeregisterStatusCallback
  - vehicle-state.h, [57](#)
- snsVehicleStateDestroy
  - vehicle-state.h, [57](#)
- snsVehicleStateGetMetaData
  - vehicle-state.h, [57](#)
- snsVehicleStateGetStatus
  - vehicle-state.h, [57](#)
- snsVehicleStateGetVehicleStateData
  - vehicle-state.h, [57](#)
- snsVehicleStateInit
  - vehicle-state.h, [58](#)
- snsVehicleStateRegisterCallback
  - vehicle-state.h, [58](#)
- snsVehicleStateRegisterStatusCallback
  - vehicle-state.h, [58](#)
- snsWheelDeregisterCallback
  - wheel.h, [62](#)
- snsWheelDeregisterStatusCallback
  - wheel.h, [63](#)
- snsWheelDestroy
  - wheel.h, [63](#)
- snsWheelGetConfiguration
  - wheel.h, [63](#)
- snsWheelGetMetaData
  - wheel.h, [63](#)
- snsWheelGetStatus
  - wheel.h, [63](#)
- snsWheelGetWheelData
  - wheel.h, [64](#)
- snsWheelInit
  - wheel.h, [64](#)
- snsWheelRegisterCallback
  - wheel.h, [64](#)
- snsWheelRegisterStatusCallback
  - wheel.h, [64](#)
- status
  - TIInclinationData, [8](#)
  - TSensorStatus, [11](#)
- statusBits
  - TWheelData, [18](#)
- steering-angle.h
  - STEERINGANGLE\_FRONT\_VALID, [47](#)
  - STEERINGANGLE\_REAR\_VALID, [47](#)
  - STEERINGANGLE\_STEERINGWHEEL\_VALID, [47](#)
- steering-angle.h, [46](#)
  - ESteeringAngleValidityBits, [47](#)
  - snsSteeringAngleDeregisterCallback, [47](#)
  - snsSteeringAngleDeregisterStatusCallback, [47](#)
  - snsSteeringAngleDestroy, [49](#)
  - snsSteeringAngleGetConfiguration, [49](#)
  - snsSteeringAngleGetMetaData, [49](#)
  - snsSteeringAngleGetStatus, [49](#)

- snsSteeringAngleGetSteeringAngleData, 49
- snsSteeringAngleInit, 50
- snsSteeringAngleRegisterCallback, 50
- snsSteeringAngleRegisterStatusCallback, 50
- SteeringAngleCallback, 47
- SteeringAngleCallback
  - steering-angle.h, 47
- steeringRatio
  - TSteeringAngleConfiguration, 13
- steeringWheel
  - TSteeringAngleData, 13
- TAccelerationConfiguration, 1
  - anglePitch, 3
  - angleRoll, 3
  - angleYaw, 3
  - dist2RefPointX, 3
  - dist2RefPointY, 3
  - dist2RefPointZ, 3
  - sigmaX, 3
  - sigmaY, 3
  - sigmaZ, 3
  - typeBits, 3
  - validityBits, 3
- TAccelerationData, 3
  - temperature, 4
  - timestamp, 4
  - validityBits, 4
  - x, 4
  - y, 4
  - z, 4
- TDistance3D, 4
  - x, 5
  - y, 5
  - z, 5
- TGyroscopeConfiguration, 5
  - anglePitch, 6
  - angleRoll, 6
  - angleYaw, 6
  - momentOfYawInertia, 6
  - sigmaGyroscope, 7
  - typeBits, 7
  - validityBits, 7
- TGyroscopeData, 7
  - pitchRate, 7
  - rollRate, 7
  - temperature, 8
  - timestamp, 8
  - validityBits, 8
  - yawRate, 8
- TInclinationData, 8
  - longitudinalGradientRoadway, 8
  - status, 8
  - timestamp, 8
  - traverseGradientRoadway, 9
  - validityBits, 9
- TOdometerData, 9
  - timestamp, 9
  - travelledDistance, 9
  - validityBits, 9
- TReverseGearData, 10
  - isReverseGear, 10
  - timestamp, 10
  - validityBits, 10
- TSensorMetaData, 10
  - category, 11
  - cycleTime, 11
  - type, 11
  - version, 11
- TSensorStatus, 11
  - status, 11
  - timestamp, 11
  - validityBits, 11
- TSlipAngleData, 12
  - slipAngle, 12
  - timestamp, 12
  - validityBits, 12
- TSteeringAngleConfiguration, 12
  - sigmaSteeringAngle, 13
  - sigmaSteeringWheelAngle, 13
  - steeringRatio, 13
- TSteeringAngleData, 13
  - front, 13
  - rear, 13
  - steeringWheel, 13
  - timestamp, 13
  - validityBits, 13
- TVehicleDataConfiguration, 14
  - distCoG2RefPoint, 14
  - distFrontAxle2RefPoint, 14
  - distRearAxle2RefPoint, 14
  - drivenAxles, 14
  - frontAxleTrackWidth, 14
  - seatCount, 14
  - trackWidth, 14
  - validityBits, 15
  - vehicleMass, 15
  - vehicleType, 15
  - vehicleWidth, 15
  - wheelBase, 15
- TVehicleSpeedData, 15
  - timestamp, 15
  - validityBits, 15
  - vehicleSpeed, 15
- TVehicleStateData, 16
  - antiLockBrakeSystemActive, 16
  - brakeActive, 16
  - electronicStabilityProgramActive, 16
  - timestamp, 16
  - tractionControlActive, 16
  - validityBits, 16
- TWheelConfiguration, 17
  - tireRollingCircumference, 17
  - validityBits, 17
  - wheel1Id, 17
  - wheel2Id, 17
  - wheel3Id, 17



- wheel4Id, 17
- wheelUnit, 17
- wheelTicksPerRevolution, 17
- TWheelData, 18
  - statusBits, 18
  - timestamp, 18
  - validityBits, 18
  - wheel1, 18
  - wheel2, 18
  - wheel3, 18
  - wheel4, 18
- temperature
  - TAccelerationData, 4
  - TGyroscopeData, 8
- timestamp
  - TAccelerationData, 4
  - TGyroscopeData, 8
  - TInclinationData, 8
  - TOdometerData, 9
  - TReverseGearData, 10
  - TSensorStatus, 11
  - TSlipAngleData, 12
  - TSteeringAngleData, 13
  - TVehicleSpeedData, 15
  - TVehicleStateData, 16
  - TWheelData, 18
- tireRollingCircumference
  - TWheelConfiguration, 17
- trackWidth
  - TVehicleDataConfiguration, 14
- tractionControlActive
  - TVehicleStateData, 16
- travelledDistance
  - TOdometerData, 9
- traverseGradientRoadway
  - TInclinationData, 9
- type
  - TSensorMetaData, 11
- typeBits
  - TAccelerationConfiguration, 3
  - TGyroscopeConfiguration, 7
- VEHICLESPEED\_VEHICLESPEED\_VALID
  - vehicle-speed.h, 53
- VEHICLESTATE\_ANTILOCKBRAKESYSTEMACTIVE\_VALID
  - vehicle-state.h, 56
- VEHICLESTATE\_BRAKEACTIVE\_VALID
  - vehicle-state.h, 56
- VEHICLESTATE\_ELECTRONICSTABILITYPROGRAMACTIVE\_VALID
  - vehicle-state.h, 56
- VEHICLESTATE\_TRACTIONCONTROLACTIVE\_VALID
  - vehicle-state.h, 56
- VEHICLESTATUS\_DISTCOG2REFPOINT\_VALID
  - vehicle-data.h, 51
- VEHICLESTATUS\_DISTFRONTAXLE2REFPOINT\_VALID
  - vehicle-data.h, 51
- VEHICLESTATUS\_DISTREARAXLE2REFPOINT\_VALID
  - vehicle-data.h, 52
- VEHICLESTATUS\_DRIVENAXLES\_VALID
  - vehicle-data.h, 51
- VEHICLESTATUS\_FRONTAXLETRACKWIDTH\_VALID
  - vehicle-data.h, 51
- VEHICLESTATUS\_SEATCOUNT\_VALID
  - vehicle-data.h, 51
- VEHICLESTATUS\_TRACKWIDTH\_VALID
  - vehicle-data.h, 51
- VEHICLESTATUS\_VEHICLEMASS\_VALID
  - vehicle-data.h, 51
- validityBits
  - TAccelerationConfiguration, 3
  - TAccelerationData, 4
  - TGyroscopeConfiguration, 7
  - TGyroscopeData, 8
  - TInclinationData, 9
  - TOdometerData, 9
  - TReverseGearData, 10
  - TSensorStatus, 11
  - TSlipAngleData, 12
  - TSteeringAngleData, 13
  - TVehicleDataConfiguration, 15
  - TVehicleSpeedData, 15
  - TVehicleStateData, 16
  - TWheelConfiguration, 17
  - TWheelData, 18
- vehicle-data.h
  - SNS\_AXLE\_BOTH, 51
  - SNS\_AXLE\_FRONT, 51
  - SNS\_AXLE\_REAR, 51
  - SNS\_AXLE\_UNKNOWN, 51
  - SNS\_BUS, 52
  - SNS\_CAR, 52
  - SNS\_MOTORBIKE, 52
  - SNS\_TRUCK, 52
  - VEHICLESTATUS\_DISTCOG2REFPOINT\_VALID, 51
  - VEHICLESTATUS\_DISTFRONTAXLE2REFPOINT\_VALID, 51
  - VEHICLESTATUS\_DISTREARAXLE2REFPOINT\_VALID, 52
  - VEHICLESTATUS\_DRIVENAXLES\_VALID, 51
  - VEHICLESTATUS\_FRONTAXLETRACKWIDTH\_VALID, 51
  - VEHICLESTATUS\_SEATCOUNT\_VALID, 51
  - VEHICLESTATUS\_TRACKWIDTH\_VALID, 51
  - VEHICLESTATUS\_VEHICLEMASS\_VALID, 51

- VEHICLESTATUS\_VEHICLETYPE\_VALID, 51
- VEHICLESTATUS\_VEHICLEWIDTH\_VALID, 51
- VEHICLESTATUS\_WHELLBASE\_VALID, 51
- vehicle-data.h, 50
  - EAxleType, 51
  - EVehicleDataConfigurationValidityBits, 51
  - EVehicleType, 52
  - snsVehicleDataDestroy, 52
  - snsVehicleDataGetConfiguration, 52
  - snsVehicleDataInit, 52
- vehicle-speed.h
  - VEHICLESPEED\_VEHICLESPEED\_VALID, 53
- vehicle-speed.h, 52
  - EVehicleSpeedValidityBits, 53
  - snsVehicleSpeedDeregisterCallback, 53
  - snsVehicleSpeedDeregisterStatusCallback, 54
  - snsVehicleSpeedDestroy, 54
  - snsVehicleSpeedGetMetaData, 54
  - snsVehicleSpeedGetStatus, 54
  - snsVehicleSpeedGetVehicleSpeedData, 54
  - snsVehicleSpeedInit, 55
  - snsVehicleSpeedRegisterCallback, 55
  - snsVehicleSpeedRegisterStatusCallback, 55
  - VehicleSpeedCallback, 53
- vehicle-state.h
  - VEHICLESTATE\_ANTILOCKBRAKESYSTEMACTIVE\_VALID, 56
  - VEHICLESTATE\_BRAKEACTIVE\_VALID, 56
  - VEHICLESTATE\_ELECTRONICSTABILITYPROGRAMACTIVE\_VALID, 56
  - VEHICLESTATE\_TRACTIONCONTROLACTIVE\_VALID, 56
- vehicle-state.h, 55
  - EVehicleStateValidityBits, 56
  - snsVehicleStateDeregisterCallback, 56
  - snsVehicleStateDeregisterStatusCallback, 57
  - snsVehicleStateDestroy, 57
  - snsVehicleStateGetMetaData, 57
  - snsVehicleStateGetStatus, 57
  - snsVehicleStateGetVehicleStateData, 57
  - snsVehicleStateInit, 58
  - snsVehicleStateRegisterCallback, 58
  - snsVehicleStateRegisterStatusCallback, 58
  - VehicleStateCallback, 56
- vehicleMass
  - TVehicleDataConfiguration, 15
- vehicleSpeed
  - TVehicleSpeedData, 15
- VehicleSpeedCallback
  - vehicle-speed.h, 53
- VehicleStateCallback
  - vehicle-state.h, 56
- vehicleType
  - TVehicleDataConfiguration, 15
- vehicleWidth
  - TVehicleDataConfiguration, 15
- version
  - TSensorMetaData, 11
- WHEEL1\_VALID
  - wheel.h, 61
- WHEEL2\_VALID
  - wheel.h, 61
- WHEEL3\_VALID
  - wheel.h, 61
- WHEEL4\_VALID
  - wheel.h, 61
- WHEEL\_AXLE\_FRONT
  - wheel.h, 60
- WHEEL\_AXLE\_NONDRIVEN
  - wheel.h, 60
- WHEEL\_AXLE\_REAR
  - wheel.h, 60
- WHEEL\_CONFIG\_TICKS\_PER\_REV
  - wheel.h, 60
- WHEEL\_CONFIG\_TIRE\_CIRC
  - wheel.h, 60
- WHEEL\_INVALID
  - wheel.h, 60
- WHEEL\_LEFT\_FRONT
  - wheel.h, 60
- WHEEL\_LEFT\_REAR
  - wheel.h, 60
- WHEEL\_RIGHT\_FRONT
  - wheel.h, 60
- WHEEL\_RIGHT\_REAR
  - wheel.h, 60
- WHEEL\_STATUS\_GAP
  - wheel.h, 61
- WHEEL\_STATUS\_INIT
  - wheel.h, 61
- WHEEL\_UNIT\_ANGULAR\_SPEED
  - wheel.h, 61
- WHEEL\_UNIT\_SPEED
  - wheel.h, 61
- WHEEL\_UNIT\_TICKS
  - wheel.h, 61
- WHEEL\_UNKNOWN
  - wheel.h, 60
- wheel.h
  - WHEEL1\_VALID, 61
  - WHEEL2\_VALID, 61
  - WHEEL3\_VALID, 61
  - WHEEL4\_VALID, 61
  - WHEEL\_AXLE\_FRONT, 60
  - WHEEL\_AXLE\_NONDRIVEN, 60
  - WHEEL\_AXLE\_REAR, 60
  - WHEEL\_CONFIG\_TICKS\_PER\_REV, 60
  - WHEEL\_CONFIG\_TIRE\_CIRC, 60
  - WHEEL\_INVALID, 60
  - WHEEL\_LEFT\_FRONT, 60
  - WHEEL\_LEFT\_REAR, 60
  - WHEEL\_RIGHT\_FRONT, 60
  - WHEEL\_RIGHT\_REAR, 60
  - WHEEL\_STATUS\_GAP, 61
  - WHEEL\_STATUS\_INIT, 61
  - WHEEL\_UNIT\_ANGULAR\_SPEED, 61



- WHEEL\_UNIT\_SPEED, [61](#)
- WHEEL\_UNIT\_TICKS, [61](#)
- WHEEL\_UNKNOWN, [60](#)
- wheel.h, [58](#)
  - EWheelConfigValidityBits, [60](#)
  - EWheelId, [60](#)
  - EWheelStatusBits, [60](#)
  - EWheelUnit, [61](#)
  - EWheelValidityBits, [61](#)
  - snsWheelDeregisterCallback, [62](#)
  - snsWheelDeregisterStatusCallback, [63](#)
  - snsWheelDestroy, [63](#)
  - snsWheelGetConfiguration, [63](#)
  - snsWheelGetMetaData, [63](#)
  - snsWheelGetStatus, [63](#)
  - snsWheelGetWheelData, [64](#)
  - snsWheelInit, [64](#)
  - snsWheelRegisterCallback, [64](#)
  - snsWheelRegisterStatusCallback, [64](#)
  - WheelCallback, [59](#)
  - WheelStatusCallback, [59](#)
- wheel1
  - TWheelData, [18](#)
- wheel1Id
  - TWheelConfiguration, [17](#)
- wheel2
  - TWheelData, [18](#)
- wheel2Id
  - TWheelConfiguration, [17](#)
- wheel3
  - TWheelData, [18](#)
- wheel3Id
  - TWheelConfiguration, [17](#)
- wheel4
  - TWheelData, [18](#)
- wheel4Id
  - TWheelConfiguration, [17](#)
- wheelBase
  - TVehicleDataConfiguration, [15](#)
- WheelCallback
  - wheel.h, [59](#)
- WheelStatusCallback
  - wheel.h, [59](#)
- wheelUnit
  - TWheelConfiguration, [17](#)
- wheelticksPerRevolution
  - TWheelConfiguration, [17](#)
- x
  - TAccelerationData, [4](#)
  - TDistance3D, [5](#)
- y
  - TAccelerationData, [4](#)
  - TDistance3D, [5](#)
- yawRate
  - TGyroscopeData, [8](#)
- z