



## **GENIVI SensorsService**

# **Component Specification**

Release 3.0.0  
Status: Released  
20.01.2015

**Accepted for release by:**

This document has is a draft of the SensorsService API 3.0.0 defined by the GENIVI expert group Location Based Services (LBS).

**Abstract:**

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

**Keywords:**

SensorsService, Sensors, Positioning API.

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

Copyright (C) 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. <i>Context</i> .....	20
2. Context Diagram .....	20



## Change History

<b>Version</b>	<b>Date</b>	<b>Author</b>	<b>Change</b>
0.1	27.08.2013	MResidori	Document Created
0.2	18.11.2013	MResidori	Document generated from the Entrprise Architect Model
0.3	27.03.2014	MResidori	Added copyright notes
3.0.0-alpha	24.04.2014	MResidori	Changed license version from 3.0 to 4.0
3.0.0-alpha	10.12.2014	MResidori	Updated API description
3.0.0-alpha	20.01.2015	HSchmidt	Fix copy/paste error
3.0.0	20.01.2015	MResidori	Changed status to “Released” (after System Architecture Team approval)

# 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



Figure: 1

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

Forward a Set of Sensor Values	
«GFunctionalRequirement»	Priority: Medium
<p><b>Description:</b> The Enhanced Position contains in addition to the Position and Course values as well a set of sensor data.</p> <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> <p><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.</p>	

Provides Data via IPC	
«GFunctionalRequirement»	Priority: Medium
<p><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.</p> <p><b>Rational:</b> Several SW components in the system are clients for the result of the filtered position and need to access the data.</p>	

Support of different Global Navigation Satellite Systems (GNSS) to calculate the current position.	
«GFunctionalRequirement»	Priority: Medium
<p>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).</p>	

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
<p><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:</p> <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> <p><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.</p>	

Extended Acceleration Sensor	
«GFunctionalRequirement»	Priority: Low
<p><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.</p> <p><b>Rational:</b>            Used for optimizing the dead reckoning solution.</p>	

Extended GNSS Service	
«GFunctionalRequirement»	Priority: Medium
<p><b>Description:</b>            The software platform provides an extension to the GNSS Service with optional information.</p> <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<sup>-7</sup>)

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
<p><b>Description:</b>            The software platform provides the following information about the Sensor and the related output Signals:</p> <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> <p><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.</p>	

Sensor Signal Timestamp	
«GFunctionalRequirement»	Priority: High
<p><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.</p> <p><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.</p>	

Signal Measurement Units	
«GFunctionalRequirement»	Priority: High
<p><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.</p> <p><b>Rational:</b>            This decouples the client applications from the implementation details of individual sensor devices.</p>	

Signal Values Type Compatibility	
«GFunctionalRequirement»	Priority: Medium
<p><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.</p> <p><b>Rational:</b> Sensor service clients are able to use multiple Sensor Services without changes in the interfaces.</p>	

Simple Gyroscope Sensor Service	
«GFunctionalRequirement»	Priority: Medium
<p><b>Description:</b> The software platform includes the Sensor that delivers</p> <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> <p>at the frequency of at least 5Hz. Unit of yaw rate is "degrees per second".</p> <p>Sign of yaw rate is defined by rule of right hand (thumb direction: up) (see reference system)</p> <p><b>Rational:</b> This Sensor Service is used in Dead Reckoning calculations of the vehicle position.</p>	

Slip Angle Sensor	
«GFunctionalRequirement»	Priority: Low
<p><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).</p> <p><b>Rational:</b> This Sensor is used for optimizations in Dead Reckoning calculations of the vehicle position.</p>	

SteeringAngle Sensor	
«GFunctionalRequirement»	Priority: Low
<p><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.</p> <p><b>Rational:</b> Is used as additional element for plausibilisation of the yaw rate in the dead reckoning module.</p>	



Vehicle State Sensor	
«GFunctionalRequirement»	Priority: Low
<p><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</p> <p><b>Rational:</b>  This Sensor is used for optimizations in Dead Reckoning calculations of the vehicle position.</p>	

VehicleSpeed Sensor	
«GFunctionalRequirement»	Priority: Medium
<p><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.</p> <p><b>Rational:</b>  Vehicle speed is sometimes the only speed related signal available to the head unit.</p>	

Wheel Tick/Speed Sensor Service	
«GFunctionalRequirement»	Priority: Medium
<p><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).</p> <p>The resolution of a single wheel revolution (i.e. the number of ticks per revolution) is included with the Sensor Service meta-data.</p> <p>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].</p> <p><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.</p>	

## 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

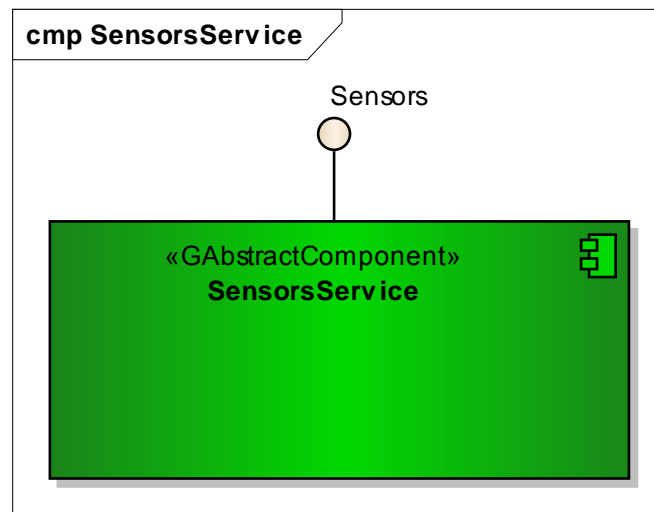


Figure: 2

### 3. Traceability Diagram

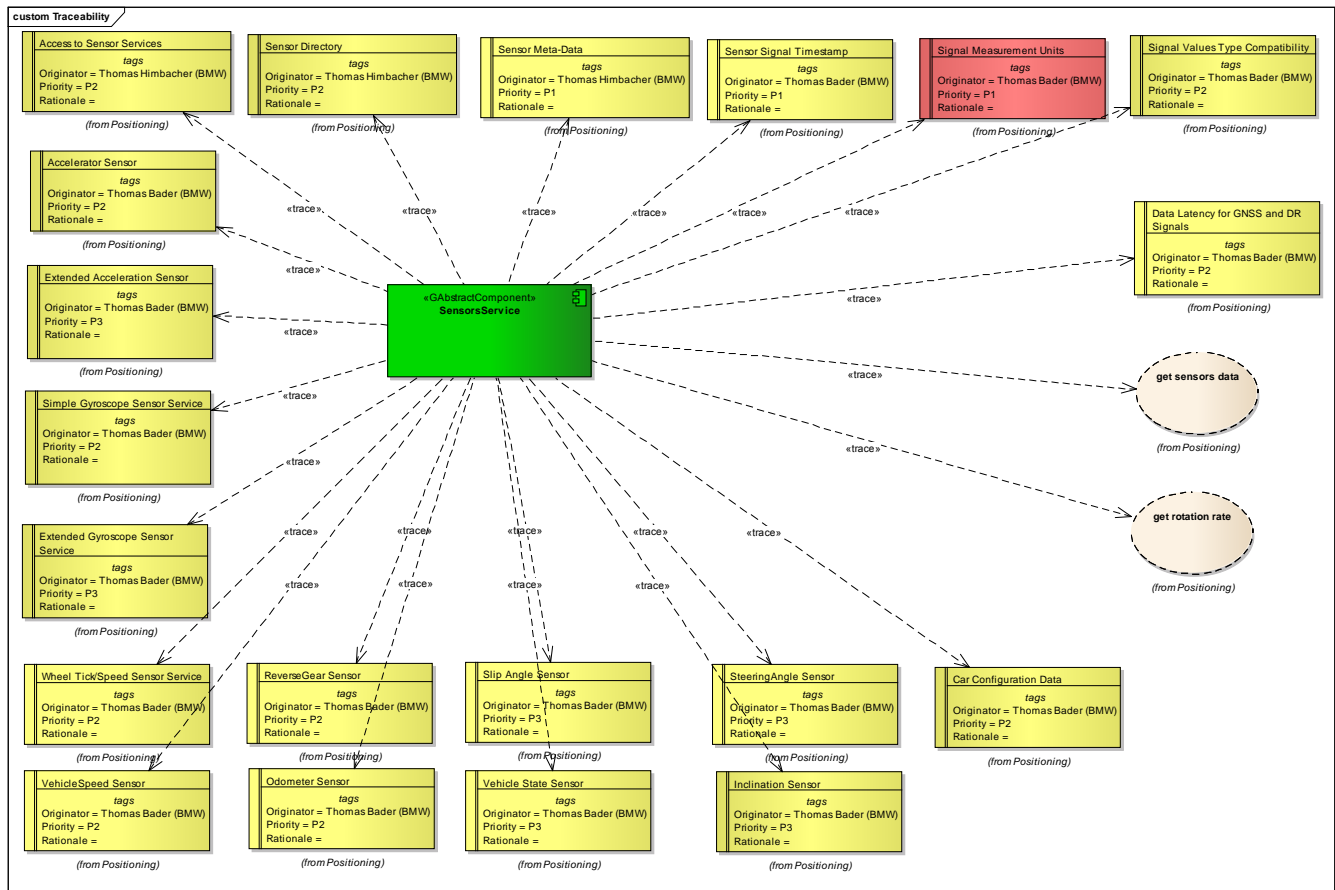


Figure: 3

## 1. Context

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

## 2. Context Diagram

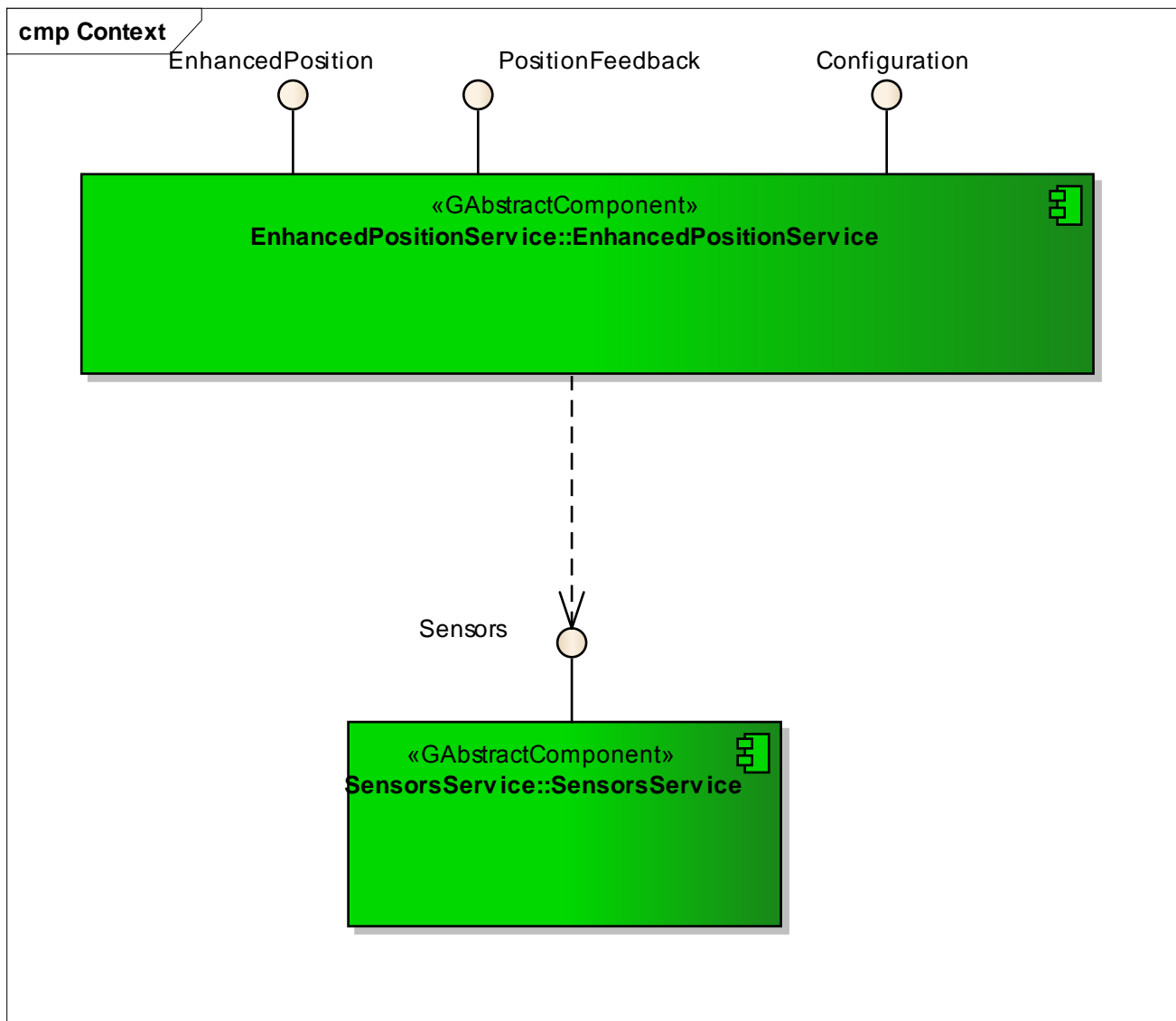


Figure: 4

# SensorsService

Generated by Doxygen 1.8.6

Wed Jan 21 2015 14:23:43

## Contents

<b>1</b>	<b>Class Documentation</b>	<b>1</b>
1.1	TAccelerationConfiguration Struct Reference	2
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	5
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	9
1.8.1	Detailed Description	9
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	10
1.10	TSlipAngleData Struct Reference	11
1.10.1	Detailed Description	11
1.10.2	Member Data Documentation	11
1.11	TSteeringAngleConfiguration Struct Reference	11
1.11.1	Detailed Description	11
1.11.2	Member Data Documentation	12
1.12	TSteeringAngleData Struct Reference	12
1.12.1	Detailed Description	12
1.12.2	Member Data Documentation	12
1.13	TVehicleSpeedData Struct Reference	13

1.13.1	Detailed Description	13
1.13.2	Member Data Documentation	13
1.14	TVehicleStateData Struct Reference	13
1.14.1	Detailed Description	14
1.14.2	Member Data Documentation	14
1.15	TWheelspeed Struct Reference	14
1.15.1	Detailed Description	14
1.15.2	Member Data Documentation	14
1.16	TWheelspeedAngular Struct Reference	15
1.16.1	Detailed Description	15
1.16.2	Member Data Documentation	15
1.17	TWheelspeedAngularDetail Struct Reference	15
1.17.1	Detailed Description	15
1.17.2	Member Data Documentation	16
1.18	TWheelspeedDetail Struct Reference	16
1.18.1	Detailed Description	16
1.18.2	Member Data Documentation	16
1.19	TWheeltickDetail Struct Reference	16
1.19.1	Detailed Description	16
1.19.2	Member Data Documentation	17
1.20	TWheelticks Struct Reference	17
1.20.1	Detailed Description	17
1.20.2	Member Data Documentation	17
<b>2</b>	<b>File Documentation</b>	<b>17</b>
2.1	acceleration.h File Reference	17
2.1.1	Typedef Documentation	18
2.1.2	Enumeration Type Documentation	19
2.1.3	Function Documentation	19
2.2	gyroscope.h File Reference	21
2.2.1	Typedef Documentation	22
2.2.2	Enumeration Type Documentation	22
2.2.3	Function Documentation	23
2.3	inclination.h File Reference	24
2.3.1	Typedef Documentation	25
2.3.2	Enumeration Type Documentation	25
2.3.3	Function Documentation	26
2.4	odometer.h File Reference	27
2.4.1	Typedef Documentation	27
2.4.2	Enumeration Type Documentation	27

2.4.3	Function Documentation	28
2.5	reverse-gear.h File Reference	29
2.5.1	Typedef Documentation	29
2.5.2	Enumeration Type Documentation	30
2.5.3	Function Documentation	30
2.6	slip-angle.h File Reference	31
2.6.1	Typedef Documentation	32
2.6.2	Enumeration Type Documentation	32
2.6.3	Function Documentation	32
2.7	sns-init.h File Reference	33
2.7.1	Macro Definition Documentation	34
2.7.2	Function Documentation	34
2.8	sns-meta-data.h File Reference	34
2.8.1	Enumeration Type Documentation	35
2.8.2	Function Documentation	35
2.9	steering-angle.h File Reference	36
2.9.1	Typedef Documentation	36
2.9.2	Enumeration Type Documentation	37
2.9.3	Function Documentation	37
2.10	vehicle-data.h File Reference	38
2.10.1	Enumeration Type Documentation	39
2.10.2	Function Documentation	39
2.11	vehicle-speed.h File Reference	41
2.11.1	Typedef Documentation	42
2.11.2	Enumeration Type Documentation	42
2.11.3	Function Documentation	42
2.12	vehicle-state.h File Reference	44
2.12.1	Typedef Documentation	44
2.12.2	Enumeration Type Documentation	45
2.12.3	Function Documentation	45
2.13	wheel.h File Reference	46
2.13.1	Macro Definition Documentation	47
2.13.2	Typedef Documentation	47
2.13.3	Enumeration Type Documentation	48
2.13.4	Function Documentation	48
<b>Index</b>		<b>54</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](#)
- [EAccelerationTypeBits](#) [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 ( $X_a$ ,  $Y_a$ ,  $Z_a$ ) with respect to the vehicle axis system ( $X_v$ ,  $Y_v$ ,  $Z_v$ ) 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  $Z_v$  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 ( $X_v$ ,  $Y_v$ ,  $Z_v$ ) are oriented as follows

- $X_v$  is in the horizontal plane, pointing forwards
- $Y_v$  is in the horizontal plane, pointing to the left
- $Z_v$  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 in 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 EAccelerationTypeBits TAccelerationConfiguration::typeBits

Bit mask indicating the type of the used gyroscope.

#### 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 [m/s<sup>2</sup>].

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

The acceleration in direction of the Y-axis of the accelerometer sensor [m/s<sup>2</sup>].

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

The acceleration in direction of the Z-axis of the accelerometer sensor [m/s<sup>2</sup>].

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.

#### 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](#)
- [EGyroscopeTypeBits](#) 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 EGyroscopeTypeBits TGyroscopeConfiguration::typeBits

Bit mask indicating the type of the used gyroscope.

#### 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. Preferrably 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. Preferrably 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 TInclinationData::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 TInclinationData::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 TInclinationData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed [EInclinationValidityBits](#) 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 TSlipAngleData Struct Reference

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

### Public Attributes

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

#### 1.10.1 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.

#### 1.10.2 Member Data Documentation

##### 1.10.2.1 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.

##### 1.10.2.2 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.

##### 1.10.2.3 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](#)

## 1.11 TSteeringAngleConfiguration Struct Reference

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

### Public Attributes

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

#### 1.11.1 Detailed Description

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

### 1.11.2 Member Data Documentation

#### 1.11.2.1 float TSteeringAngleConfiguration::sigmaSteeringAngle

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

#### 1.11.2.2 float TSteeringAngleConfiguration::sigmaSteeringWheelAngle

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

#### 1.11.2.3 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](#)

## 1.12 TSteeringAngleData Struct Reference

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

### Public Attributes

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

### 1.12.1 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.

### 1.12.2 Member Data Documentation

#### 1.12.2.1 float TSteeringAngleData::front

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

#### 1.12.2.2 float TSteeringAngleData::rear

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

#### 1.12.2.3 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](#).

#### 1.12.2.4 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.

#### 1.12.2.5 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](#)

## 1.13 TVehicleSpeedData Struct Reference

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

### Public Attributes

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

#### 1.13.1 Detailed Description

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

#### 1.13.2 Member Data Documentation

##### 1.13.2.1 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.

##### 1.13.2.2 uint32\_t TVehicleSpeedData::validityBits

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

##### 1.13.2.3 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](#)

## 1.14 TVehicleStateData Struct Reference

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

### Public Attributes

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

### 1.14.1 Detailed Description

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

### 1.14.2 Member Data Documentation

#### 1.14.2.1 `bool TVehicleStateData::antiLockBrakeSystemActive`

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

#### 1.14.2.2 `bool TVehicleStateData::brakeActive`

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

#### 1.14.2.3 `bool TVehicleStateData::electronicStabilityProgramActive`

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

#### 1.14.2.4 `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.

#### 1.14.2.5 `bool TVehicleStateData::tractionControlActive`

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

#### 1.14.2.6 `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](#)

## 1.15 TWheelspeed Struct Reference

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

### Public Attributes

- `uint64_t timestamp`
- `TWheelspeedDetail elements [WHEEL_NUM_ELEMENTS]`

### 1.15.1 Detailed Description

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

### 1.15.2 Member Data Documentation

#### 1.15.2.1 `TWheelspeedDetail TWheelspeed::elements[WHEEL_NUM_ELEMENTS]`

The array of wheel speed elements.

### 1.15.2.2 uint64\_t TWheelspeed::timestamp

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

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

- [wheel.h](#)

## 1.16 TWheelspeedAngular Struct Reference

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

### Public Attributes

- [uint64\\_t timestamp](#)
- [TWheelspeedAngularDetail elements \[WHEEL\\_NUM\\_ELEMENTS\]](#)

### 1.16.1 Detailed Description

Angular wheel speed information for multiple wheels. Container with timestamp as used by callback and get function.

### 1.16.2 Member Data Documentation

#### 1.16.2.1 TWheelspeedAngularDetail TWheelspeedAngular::elements[WHEEL\_NUM\_ELEMENTS]

The array of angular wheel speed elements.

#### 1.16.2.2 uint64\_t TWheelspeedAngular::timestamp

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

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

- [wheel.h](#)

## 1.17 TWheelspeedAngularDetail Struct Reference

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

### Public Attributes

- [EWheelId id](#)
- float [rotations](#)

### 1.17.1 Detailed Description

A single angular wheel speed information. No valid flags are provided for the angular wheel speeds as this information is transferred as an array. If data is not valid the data is simply omitted from transfer.

### 1.17.2 Member Data Documentation

#### 1.17.2.1 EWheelId TWheelspeedAngularDetail::id

#### 1.17.2.2 float TWheelspeedAngularDetail::rotations

unit is [1/s] rotation per seconds

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

- [wheel.h](#)

## 1.18 TWheelspeedDetail Struct Reference

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

### Public Attributes

- [EWheelId id](#)
- float [speedOfWheel](#)

### 1.18.1 Detailed Description

A single wheel speed information. No valid flags are provided for the wheel speeds as this information is transferred as an array. If data is not valid the data is simply omitted from transfer.

### 1.18.2 Member Data Documentation

#### 1.18.2.1 EWheelId TWheelspeedDetail::id

#### 1.18.2.2 float TWheelspeedDetail::speedOfWheel

[m/s]

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

- [wheel.h](#)

## 1.19 TWheeltickDetail Struct Reference

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

### Public Attributes

- [EWheelId wheeltickIdentifier](#)
- uint32\_t [wheeltickCounter](#)

### 1.19.1 Detailed Description

A single wheel tick information. No valid flags are provided for the wheel ticks as this information is transferred as an array. If data is not valid the data is simply omitted from transfer.

### 1.19.2 Member Data Documentation

#### 1.19.2.1 uint32\_t TWheeltickDetail::wheeltickCounter

The actual wheel tick counter which provides the ticks for the type specified in wheeltickIdentifier. This is a running counter which overflows only at a vehicle specific threshold. This must be tracked by the client. The threshold value can be determined by calling [snsWheeltickGetWheelticksCountMax\(\)](#)

#### 1.19.2.2 EWheelId TWheeltickDetail::wheeltickIdentifier

Type for which this tick information is provided.

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

- [wheel.h](#)

## 1.20 TWheelticks Struct Reference

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

### Public Attributes

- [uint64\\_t timestamp](#)
- [TWheeltickDetail elements \[WHEEL\\_NUM\\_ELEMENTS\]](#)

### 1.20.1 Detailed Description

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

### 1.20.2 Member Data Documentation

#### 1.20.2.1 TWheeltickDetail TWheelticks::elements[WHEEL\_NUM\_ELEMENTS]

The array of wheeltick elements.

#### 1.20.2.2 uint64\_t TWheelticks::timestamp

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

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 <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)

## 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: <code>&gt;=1</code> . 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::sigmaY](#).  
**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 EGYroscopeTypeBits 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 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.

**2.1.3.3 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

**2.1.3.4 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

**2.1.3.5 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.

## 2.1.3.6 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.

2.1.3.7 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.

## 2.2 gyroscope.h File Reference

```
#include "sns-meta-data.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)

### 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 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.

## 2.2.3.3 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

## 2.2.3.4 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

**2.2.3.5 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.

**2.2.3.6 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.

**2.2.3.7 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.

**2.3 inclination.h File Reference**

```
#include "sns-meta-data.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\\_C-  
CONDITION](#), [INCLINATION\\_MODEL\\_BASED](#),  
[INCLINATION\\_NOT\\_AVAILABLE](#) }
- enum [EInclinationValidityBits](#) { [INCLINATION\\_LONGITUDINAL\\_VALID](#) = 0x00000001, [INCLINATION\\_TRA-  
VERSE\\_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)

## 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: >=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 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.

#### 2.3.3.3 `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

#### 2.3.3.4 `bool snsInclinationGetMetaData ( 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.

#### 2.3.3.5 `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.

#### 2.3.3.6 `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.

## 2.4 odometer.h File Reference

```
#include "sns-meta-data.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)

## 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 numElements
<i>numElements</i>	allowed range: $\geq 1$ . If numElements $> 1$ , 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 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.

2.4.3.3 `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.

2.4.3.4 `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

2.4.3.5 `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.

**2.4.3.6 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.

**2.5 reverse-gear.h File Reference**

```
#include "sns-meta-data.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)

**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 numElements $> 1$ , buffered data are provided.

## 2.5.2 Enumeration Type Documentation

## 2.5.2.1 enum EReverseGearValidityBits

[TReverseGearData::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 [TReverseGearData::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 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.

2.5.3.3 bool snsReverseGearGetMetaData ( [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.

2.5.3.4 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

## 2.5.3.5 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.

## 2.5.3.6 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.

## 2.6 slip-angle.h File Reference

```
#include "sns-meta-data.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)

## 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 <code>numElements</code>
<i>numElements</i>	allowed range: $\geq 1$ . If <code>numElements &gt; 1</code> , buffered data are provided.

## 2.6.2 Enumeration Type Documentation

### 2.6.2.1 enum ESlipAngleValidityBits

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

#### Enumerator

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

## 2.6.3 Function Documentation

### 2.6.3.1 bool snsSlipAngleDeregisterCallback ( SlipAngleCallback callback )

Deregister slip angle callback. After calling this method no new slip 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 snsSlipAngleDestroy ( )

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

#### Returns

True if shutdown has been successful.

### 2.6.3.3 bool snsSlipAngleGetMetaData ( 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.

**2.6.3.4 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

**2.6.3.5 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.

**2.6.3.6 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.

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

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

**Macros**

- #define [GENIVI\\_SNS\\_API\\_MAJOR](#) 3
- #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 3`

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](#), [SEN-](#)

```

    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\_WHEELTICK** 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 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 steering-angle.h File Reference

```
#include "sns-meta-data.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)

### 2.9.1 Typedef Documentation

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

## 2.9.2 Enumeration Type Documentation

### 2.9.2.1 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](#).

## 2.9.3 Function Documentation

### 2.9.3.1 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.

### 2.9.3.2 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.

### 2.9.3.3 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 <i>config</i> .
---------------	---

#### Returns

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

### 2.9.3.4 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.

**2.9.3.5 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 <i>angleData</i> .
------------------	---

**Returns**

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

**2.9.3.6 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.

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

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

**Classes**

- struct [TDistance3D](#)

**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 }

## Functions

- bool `vehicleDataGetVehicleType` (`EVehicleType *type`)
- bool `vehicleDataGetDrivenAxles` (`EAxleType *axles`)
- bool `vehicleDataGetSeatCount` (`uint8_t *seatCount`)
- bool `vehicleDataGetTrackWidth` (`float *width`)
- bool `vehicleDataGetFrontAxleTrackWidth` (`float *width`)
- bool `vehicleDataGetWheelBase` (`float *wheelbase`)
- bool `vehicleDataGetVehicleMass` (`float *mass`)
- bool `vehicleDataGetVehicleWidth` (`float *width`)
- bool `vehicleDataGetDistCoG2RefPoint` (`TDistance3D *dist`)
- bool `vehicleDataGetDistFrontAxle2RefPoint` (`TDistance3D *dist`)
- bool `vehicleDataGetDistRearAxle2RefPoint` (`TDistance3D *dist`)

### 2.10.1 Enumeration Type Documentation

#### 2.10.1.1 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).

#### 2.10.1.2 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.

### 2.10.2 Function Documentation

#### 2.10.2.1 bool `vehicleDataGetDistCoG2RefPoint` (`TDistance3D * dist`)

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

##### Parameters

<i>dist</i>	The distance result in 3 dimensions is written to this parameter.
-------------	---

##### Returns

True if the configuration value is available. If false no value could be provided.

#### 2.10.2.2 bool `vehicleDataGetDistFrontAxle2RefPoint` (`TDistance3D * dist`)

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

## Parameters

<i>dist</i>	The distance result in 3 dimensions is written to this parameter.
-------------	---

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.3 `bool vehicleDataGetDistRearAxle2RefPoint ( TDistance3D * dist )`

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

## Parameters

<i>dist</i>	The distance result in 3 dimensions is written to this parameter.
-------------	---

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.4 `bool vehicleDataGetDrivenAxles ( EAxleType * axles )`

Type of the driven axles of the vehicle as provided in the official documents. This is a static configuration.

## Parameters

<i>axles</i>	The driven axles type is written to this parameter as a result as defined in the enumeration.
--------------	---

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.5 `bool vehicleDataGetFrontAxleTrackWidth ( float * width )`

Front axle track width of the vehicle as provided in the official documents. This is a static configuration. Unit: [m].

## Parameters

<i>width</i>	The vehicle track width of the front axle in m is written to this parameter as a result.
--------------	--

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.6 `bool vehicleDataGetSeatCount ( uint8_t * seatCount )`

Number of seats of the vehicle as provided in the official documents. This is a static configuration.

## Parameters

<i>seatCount</i>	The number of seats is written to this parameter as a result.
------------------	---

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.7 `bool vehicleDataGetTrackWidth ( float * width )`

Track width of the vehicle as provided in the official documents. This is a static configuration. Unit: [m]. If the vehicle has different track widths at the front and rear axles, the rear axle track is referred to.

## Parameters

<i>width</i>	The vehicle track width in m is written to this parameter as a result.
--------------	--

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.8 bool vehicleDataGetVehicleMass ( float \* *mass* )

Mass of the vehicle as provided in the official documents. This is a static configuration value and does not include the current load. Unit: [kg].

## Parameters

<i>mass</i>	The vehicle mass in kg is written to this parameter as a result.
-------------	--

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.9 bool vehicleDataGetVehicleType ( EVehicleType \* *type* )

Type of the vehicle. This is a static configuration.

## Parameters

<i>type</i>	The vehicle type is written to this parameter as a result as defined in the enumeration.
-------------	--

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.10 bool vehicleDataGetVehicleWidth ( float \* *width* )

Width of the vehicle as provided in the official documents. This is a static configuration. Unit: [m].

## Parameters

<i>width</i>	The vehicle width in m is written to this parameter as a result.
--------------	--

## Returns

True if the configuration value is available. If false no value could be provided.

2.10.2.11 bool vehicleDataGetWheelBase ( float \* *wheelbase* )

Wheel base of the vehicle as provided in the official documents. This is a static configuration. Unit: [m]. The wheel base is basically the distance between the front axle and the rear axle. For an exact definition, see ISO 8855.

## Parameters

<i>width</i>	The wheel base in m is written to this parameter as a result.
--------------	---

## Returns

True if the configuration value is available. If false no value could be provided.

## 2.11 vehicle-speed.h File Reference

```
#include "sns-meta-data.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)

### 2.11.1 Typedef Documentation

#### 2.11.1.1 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.

### 2.11.2 Enumeration Type Documentation

#### 2.11.2.1 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](#).

### 2.11.3 Function Documentation

#### 2.11.3.1 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.

## 2.11.3.2 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.

## 2.11.3.3 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.

## 2.11.3.4 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 vehicleSpeed.
---------------------	---

## Returns

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

## 2.11.3.5 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.

## 2.11.3.6 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 vehicle-state.h File Reference

```
#include "sns-meta-data.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)

## 2.12.1 Typedef Documentation

2.12.1.1 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 numElements
--------------------------	--

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

## 2.12.2 Enumeration Type Documentation

### 2.12.2.1 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.

## 2.12.3 Function Documentation

### 2.12.3.1 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.

### 2.12.3.2 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.

### 2.12.3.3 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.

### 2.12.3.4 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

**2.12.3.5 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.

**2.12.3.6 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 wheel.h File Reference**

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

**Classes**

- struct [TWheeltickDetail](#)
- struct [TWheelticks](#)
- struct [TWheelspeedAngularDetail](#)
- struct [TWheelspeedAngular](#)
- struct [TWheelspeedDetail](#)
- struct [TWheelspeed](#)

**Macros**

- #define [WHEEL\\_NUM\\_ELEMENTS](#) 4

**Typedefs**

- typedef void(\* [WheeltickCallback](#) )(const [TWheelticks](#) ticks[], uint16\_t numElements)
- typedef void(\* [WheelspeedAngularCallback](#) )(const [TWheelspeedAngular](#) rotations[], uint16\_t numElements)
- typedef void(\* [WheelspeedCallback](#) )(const [TWheelspeed](#) wheelspeed[], uint16\_t numElements)

## 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_RE-`  
`AR` = 7,  
`WHEEL_RIGHT_REAR` = 8 }

## Functions

- bool `snsWheeltickInit` ()
- bool `snsWheeltickDestroy` ()
- bool `snsWheelTickGetMetaData` (`TSensorMetaData` \*data)
- bool `snsWheeltickGetWheelticksPerRevolution` (uint16\_t \*ticks)
- bool `snsWheeltickGetWheelticksCountMax` (uint32\_t \*max\_count)
- bool `snsWheeltickGetWheelticks` (`TWheelticks` \*ticks)
- bool `snsWheeltickRegisterCallback` (`WheeltickCallback` callback)
- bool `snsWheeltickDeregisterCallback` (`WheeltickCallback` callback)
- bool `snsWheelspeedAngularInit` ()
- bool `snsWheelspeedAngularDestroy` ()
- bool `snsWheelspeedAngularGetMetaData` (`TSensorMetaData` \*data)
- bool `snsWheelspeedAngularGet` (`TWheelspeedAngular` \*wsa)
- bool `snsWheelspeedAngularRegisterCallback` (`WheelspeedAngularCallback` callback)
- bool `snsWheelspeedAngularDeregisterCallback` (`WheelspeedAngularCallback` callback)
- bool `snsWheelspeedInit` ()
- bool `snsWheelspeedDestroy` ()
- bool `snsWheelspeedGetMetaData` (`TSensorMetaData` \*data)
- bool `snsWheelspeedGet` (`TWheelspeed` \*wheelspeed)
- bool `snsWheelspeedRegisterCallback` (`WheelspeedCallback` callback)
- bool `snsWheelspeedDeregisterCallback` (`WheelspeedCallback` callback)
- bool `snsWheelGetTireRollingCircumference` (float \*circumference)

### 2.13.1 Macro Definition Documentation

#### 2.13.1.1 #define WHEEL\_NUM\_ELEMENTS 4

Maximum number of wheel elements per structure. A fix value is used because a flat data structure has advantages like simple copying, indexed access.

### 2.13.2 Typedef Documentation

#### 2.13.2.1 typedef void(\* WheelspeedAngularCallback)(const `TWheelspeedAngular` rotations[], uint16\_t numElements)

Callback type for angular wheel speed sensor service. Use this type of callback if you want to register for angular wheel speed data. Wheel tick data can be provided for multiple `WheelIds`. E.g. data can be provided for all 4 wheels. 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>ticks</i>	pointer to an array of <a href="#">TWheelspeedAngular</a> with size numElements
<i>numElements</i>	allowed range: $\geq 1$ . If numElements $> 1$ , buffered data are provided.

### 2.13.2.2 typedef void(\* WheelspeedCallback)(const TWheelspeed wheelspeed[], uint16\_t numElements)

Callback type for wheel speed sensor service. Use this type of callback if you want to register for wheel speed data. Wheel tick data can be provided for multiple WheelIds. E.g. data can be provided for all 4 wheels. 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>wheelspeed</i>	pointer to an array of <a href="#">TWheelspeed</a> with size numElements
<i>numElements</i>	allowed range: $\geq 1$ . If numElements $> 1$ , buffered data are provided.

### 2.13.2.3 typedef void(\* WheeltickCallback)(const TWheelticks ticks[], uint16\_t numElements)

Callback type for wheel tick sensor service. Use this type of callback if you want to register for wheel tick data. Wheel tick data can be provided for multiple WheelIds. E.g. data can be provided for all 4 wheels. 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>ticks</i>	pointer to an array of <a href="#">TWheelticks</a> with size numElements
<i>numElements</i>	allowed range: $\geq 1$ . If numElements $> 1$ , buffered data are provided.

## 2.13.3 Enumeration Type Documentation

### 2.13.3.1 enum EWheelId

Defines the type for which the wheel tick information is provided. Usually wheel ticks are either provided for the non-driven axle or for each wheel individually.

#### Enumerator

**WHEEL\_INVALID** Wheel tick data is invalid / the field is unused.

**WHEEL\_UNKNOWN** No information available where the wheel tick data is from.

**WHEEL\_AXLE\_NONDRIVEN** Wheel tick data is an average from the non driven axle. Can be front or rear depending on the type of drivetrain.

**WHEEL\_AXLE\_FRONT** Wheel tick data is an average from the front axle.

**WHEEL\_AXLE\_REAR** Wheel tick data is an average from the rear axle.

**WHEEL\_LEFT\_FRONT** Wheel tick data is from the left front wheel.

**WHEEL\_RIGHT\_FRONT** Wheel tick data is from the right front wheel.

**WHEEL\_LEFT\_REAR** Wheel tick data is from the left rear wheel.

**WHEEL\_RIGHT\_REAR** Wheel tick data is from the right rear wheel.

## 2.13.4 Function Documentation

### 2.13.4.1 bool snsWheelGetTireRollingCircumference ( float \* circumference )

Tire rolling circumference as provided in the official documents. This is a static configuration. Unit: [m]. Static configuration value that specifies the distance travelled on the ground per a single revolution of one wheel. This may be useful for calculations based on wheel ticks or angular wheel speeds

## Parameters

<i>circumference</i>	The tire rolling circumference in m is written to this parameter as a result.
----------------------	---

## Returns

True if configuration data is available. If false no value could be provided.

2.13.4.2 bool snsWheelspeedAngularDeregisterCallback ( *WheelspeedAngularCallback callback* )

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

## Parameters

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

## Returns

True if callback has been deregistered successfully.

## 2.13.4.3 bool snsWheelspeedAngularDestroy ( )

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

## Returns

True if shutdown has been successful.

2.13.4.4 bool snsWheelspeedAngularGet ( *TWheelspeedAngular \* wsa* )

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

## Parameters

<i>wsa</i>	After calling the method the currently available angular wheel speed data is written into the array.
------------	--

## Returns

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

2.13.4.5 bool snsWheelspeedAngularGetMetaData ( *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.

## 2.13.4.6 bool snsWheelspeedAngularInit ( )

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



**Returns**

True if initialization has been successful.

**2.13.4.7 bool snsWheelspeedAngularRegisterCallback ( WheelspeedAngularCallback callback )**

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

**Parameters**

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

**Returns**

True if callback has been registered successfully.

**2.13.4.8 bool snsWheelspeedDeregisterCallback ( WheelspeedCallback callback )**

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

**Parameters**

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

**Returns**

True if callback has been deregistered successfully.

**2.13.4.9 bool snsWheelspeedDestroy ( )**

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

**Returns**

True if shutdown has been successful.

**2.13.4.10 bool snsWheelspeedGet ( TWheelspeed \* wheelspeed )**

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

**Parameters**

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

**Returns**

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

**2.13.4.11 bool snsWheelspeedGetMetaData ( 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.

## 2.13.4.12 bool snsWheelspeedInit ( )

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

## Returns

True if initialization has been successful.

2.13.4.13 bool snsWheelspeedRegisterCallback ( *WheelspeedCallback callback* )

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

## Parameters

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

## Returns

True if callback has been registered successfully.

2.13.4.14 bool snsWheeltickDeregisterCallback ( *WheeltickCallback callback* )

Deregister multiple wheel tick callback. After calling this method no new wheel tick 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.

## 2.13.4.15 bool snsWheeltickDestroy ( )

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

## Returns

True if shutdown has been successful.

2.13.4.16 bool snsWheelTickGetMetaData ( *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.

**2.13.4.17 bool snsWheeltickGetWheelticks ( TWheelticks \* ticks )**

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

**Parameters**

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

**Returns**

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

**2.13.4.18 bool snsWheeltickGetWheelticksCountMax ( uint32\_t \* max\_count )**

Maximum value of vehicle specific wheel tick rolling counters Static configuration value that specifies *after* which value the wheel tick counter will roll over. Some examples if full 15bit are used for the vehicle specific rolling wheel counters, the maximum value would be 0x7FFF = 32767 if 15bit are used but the last value 0x7FFF is reserved as SNA then the maximum value would be 0x7FFE = 32766

**Parameters**

<i>max_count</i>	The maximum value of wheel tick rolling counters
------------------	--

**Returns**

True if configuration data is available. If false no value could be provided and an overflow only at the uint32 size shall be assumed.

**2.13.4.19 bool snsWheeltickGetWheelticksPerRevolution ( uint16\_t \* ticks )**

Wheel ticks per revolution Static configuration value that specifies how many wheel ticks there are per a single revolution of one wheel.

**Parameters**

<i>ticks</i>	The number of ticks for a single revolution of one wheel
--------------	--

**Returns**

True if configuration data is available.

**2.13.4.20 bool snsWheeltickInit ( )**

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

**Returns**

True if initialization has been successful.

**2.13.4.21 bool snsWheeltickRegisterCallback ( WheeltickCallback callback )**

Register callback for multiple wheel tick information. This is the recommended method for continuously accessing the wheel tick data. The callback will be invoked when new wheel tick 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, 19
- ACCELERATION\_CONFIG\_ANGLEROLL\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_ANGLEYAW\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_DISTX\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_DISTY\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_DISTZ\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_SIGMAX\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_SIGMAY\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_SIGMAZ\_VALID
  - acceleration.h, 19
- ACCELERATION\_CONFIG\_TYPE\_VALID
  - acceleration.h, 19
- ACCELERATION\_TEMPERATURE\_PROVIDED
  - acceleration.h, 19
- ACCELERATION\_TEMPERATURE\_VALID
  - acceleration.h, 19
- ACCELERATION\_X\_PROVIDED
  - acceleration.h, 19
- ACCELERATION\_X\_VALID
  - acceleration.h, 19
- ACCELERATION\_Y\_PROVIDED
  - acceleration.h, 19
- ACCELERATION\_Y\_VALID
  - acceleration.h, 19
- ACCELERATION\_Z\_PROVIDED
  - acceleration.h, 19
- ACCELERATION\_Z\_VALID
  - acceleration.h, 19
- acceleration.h
  - ACCELERATION\_CONFIG\_ANGLEPITCH\_VALID, 19
  - ACCELERATION\_CONFIG\_ANGLEROLL\_VALID, 19
  - ACCELERATION\_CONFIG\_ANGLEYAW\_VALID, 19
  - ACCELERATION\_CONFIG\_DISTX\_VALID, 19
  - ACCELERATION\_CONFIG\_DISTY\_VALID, 19
  - ACCELERATION\_CONFIG\_DISTZ\_VALID, 19
  - ACCELERATION\_CONFIG\_SIGMAX\_VALID, 19
  - ACCELERATION\_CONFIG\_SIGMAY\_VALID, 19
  - ACCELERATION\_CONFIG\_SIGMAZ\_VALID, 19
  - ACCELERATION\_CONFIG\_TYPE\_VALID, 19
  - ACCELERATION\_TEMPERATURE\_PROVIDED, 19
  - ACCELERATION\_TEMPERATURE\_VALID, 19
  - ACCELERATION\_X\_PROVIDED, 19
  - ACCELERATION\_X\_VALID, 19
  - ACCELERATION\_Y\_PROVIDED, 19
  - ACCELERATION\_Y\_VALID, 19
  - ACCELERATION\_Z\_PROVIDED, 19
  - ACCELERATION\_Z\_VALID, 19
  - AccelerationCallback, 18
  - AccelerationCallback
    - acceleration.h, 18
  - anglePitch
    - TAccelerationConfiguration, 3
    - TGyroscopeConfiguration, 6
  - angleRoll
    - TAccelerationConfiguration, 3
    - TGyroscopeConfiguration, 6
  - angleYaw
    - TAccelerationConfiguration, 3
    - TGyroscopeConfiguration, 6
  - antiLockBrakeSystemActive
    - TVehicleStateData, 14
  - brakeActive
    - TVehicleStateData, 14
  - category
    - TSensorMetaData, 10
  - cycleTime
    - TSensorMetaData, 10
  - dist2RefPointX
    - TAccelerationConfiguration, 3
  - dist2RefPointY
    - TAccelerationConfiguration, 3
  - dist2RefPointZ
    - TAccelerationConfiguration, 3
  - EAccelerationConfigValidityBits
    - acceleration.h, 19
  - EAccelerationTypeBits
    - acceleration.h, 19
  - EAccelerationValidityBits
    - acceleration.h, 19
  - EAxleType
    - vehicle-data.h, 39
  - EGyroscopeConfigValidityBits
    - gyroscope.h, 22
  - EGyroscopeTypeBits

- gyroscope.h, [22](#)
- EGyroscopeValidityBits
  - gyroscope.h, [22](#)
- EInclinationSensorStatus
  - inclination.h, [25](#)
- EInclinationValidityBits
  - inclination.h, [25](#)
- EOdometerValidityBits
  - odometer.h, [27](#)
- ERReverseGearValidityBits
  - reverse-gear.h, [30](#)
- ESensorCategory
  - sns-meta-data.h, [35](#)
- ESensorType
  - sns-meta-data.h, [35](#)
- ESlipAngleValidityBits
  - slip-angle.h, [32](#)
- ESteeringAngleValidityBits
  - steering-angle.h, [37](#)
- EVehicleSpeedValidityBits
  - vehicle-speed.h, [42](#)
- EVehicleStateValidityBits
  - vehicle-state.h, [45](#)
- EVehicleType
  - vehicle-data.h, [39](#)
- EWheelId
  - wheel.h, [48](#)
- electronicStabilityProgramActive
  - TVehicleStateData, [14](#)
- elements
  - TWheelspeed, [14](#)
  - TWheelspeedAngular, [15](#)
  - TWheelticks, [17](#)
- front
  - TSteeringAngleData, [12](#)
- GYROSCOPE\_CONFIG\_ANGLEPITCH\_VALID
  - gyroscope.h, [22](#)
- GYROSCOPE\_CONFIG\_ANGLEROLL\_VALID
  - gyroscope.h, [22](#)
- GYROSCOPE\_CONFIG\_ANGLEYAW\_VALID
  - gyroscope.h, [22](#)
- GYROSCOPE\_CONFIG\_MOMENTYAW\_VALID
  - gyroscope.h, [22](#)
- GYROSCOPE\_CONFIG\_SIGMAGYROSCOPE\_VALID
  - gyroscope.h, [22](#)
- GYROSCOPE\_CONFIG\_TYPE\_VALID
  - gyroscope.h, [22](#)
- GYROSCOPE\_PITCHRATE\_PROVIDED
  - gyroscope.h, [22](#)
- GYROSCOPE\_PITCHRATE\_VALID
  - gyroscope.h, [23](#)
- GYROSCOPE\_ROLLRATE\_PROVIDED
  - gyroscope.h, [22](#)
- GYROSCOPE\_ROLLRATE\_VALID
  - gyroscope.h, [23](#)
- GYROSCOPE\_TEMPERATURE\_COMPENSATED
  - gyroscope.h, [22](#)
- GYROSCOPE\_TEMPERATURE\_PROVIDED
  - gyroscope.h, [22](#)
- GYROSCOPE\_TEMPERATURE\_VALID
  - gyroscope.h, [23](#)
- GYROSCOPE\_YAWRATE\_PROVIDED
  - gyroscope.h, [22](#)
- GYROSCOPE\_YAWRATE\_VALID
  - gyroscope.h, [23](#)
- getSensorMetaDataList
  - sns-meta-data.h, [35](#)
- gyroscope.h
  - GYROSCOPE\_CONFIG\_ANGLEPITCH\_VALID, [22](#)
  - GYROSCOPE\_CONFIG\_ANGLEROLL\_VALID, [22](#)
  - GYROSCOPE\_CONFIG\_ANGLEYAW\_VALID, [22](#)
  - GYROSCOPE\_CONFIG\_MOMENTYAW\_VALID, [22](#)
  - GYROSCOPE\_CONFIG\_SIGMAGYROSCOPE\_VALID, [22](#)
  - GYROSCOPE\_CONFIG\_TYPE\_VALID, [22](#)
  - GYROSCOPE\_PITCHRATE\_PROVIDED, [22](#)
  - GYROSCOPE\_PITCHRATE\_VALID, [23](#)
  - GYROSCOPE\_ROLLRATE\_PROVIDED, [22](#)
  - GYROSCOPE\_ROLLRATE\_VALID, [23](#)
  - GYROSCOPE\_TEMPERATURE\_COMPENSATED, [22](#)
  - GYROSCOPE\_TEMPERATURE\_PROVIDED, [22](#)
  - GYROSCOPE\_TEMPERATURE\_VALID, [23](#)
  - GYROSCOPE\_YAWRATE\_PROVIDED, [22](#)
  - GYROSCOPE\_YAWRATE\_VALID, [23](#)
- gyroscope.h, [21](#)
  - EGyroscopeConfigValidityBits, [22](#)
  - EGyroscopeTypeBits, [22](#)
  - EGyroscopeValidityBits, [22](#)
  - GyroscopeCallback, [22](#)
  - snsGyroscopeDeregisterCallback, [23](#)
  - snsGyroscopeDestroy, [23](#)
  - snsGyroscopeGetConfiguration, [23](#)
  - snsGyroscopeGetGyroscopeData, [23](#)
  - snsGyroscopeGetMetaData, [24](#)
  - snsGyroscopeInit, [24](#)
  - snsGyroscopeRegisterCallback, [24](#)
- GyroscopeCallback
  - gyroscope.h, [22](#)
- INCLINATION\_INITIALIZING
  - inclination.h, [25](#)
- INCLINATION\_LONGITUDINAL\_VALID
  - inclination.h, [25](#)
- INCLINATION\_MODEL\_BASED
  - inclination.h, [25](#)
- INCLINATION\_NOT\_AVAILABLE
  - inclination.h, [25](#)
- INCLINATION\_SENSOR\_BASED
  - inclination.h, [25](#)
- INCLINATION\_STATUS\_VALID
  - inclination.h, [25](#)
- INCLINATION\_TRAVERSE\_VALID
  - inclination.h, [25](#)

- INCLINATION\_UNSTABLE\_DRIVING\_CONDITION
  - inclination.h, 25
- id
  - TWheelspeedAngularDetail, 16
  - TWheelspeedDetail, 16
- inclination.h
  - INCLINATION\_INITIALIZING, 25
  - INCLINATION\_LONGITUDINAL\_VALID, 25
  - INCLINATION\_MODEL\_BASED, 25
  - INCLINATION\_NOT\_AVAILABLE, 25
  - INCLINATION\_SENSOR\_BASED, 25
  - INCLINATION\_STATUS\_VALID, 25
  - INCLINATION\_TRAVERSE\_VALID, 25
  - INCLINATION\_UNSTABLE\_DRIVING\_CONDITION, 25
- inclination.h, 24
  - EInclinationSensorStatus, 25
  - EInclinationValidityBits, 25
  - InclinationCallback, 25
  - snsInclinationDeregisterCallback, 26
  - snsInclinationDestroy, 26
  - snsInclinationGetInclinationData, 26
  - snsInclinationGetMetaData, 26
  - snsInclinationInit, 26
  - snsInclinationRegisterCallback, 26
- InclinationCallback
  - inclination.h, 25
- isReverseGear
  - TReverseGearData, 10
- longitudinalGradientRoadway
  - TInclinationData, 8
- momentOfYawInertia
  - TGyroscopeConfiguration, 6
- ODOMETER\_TRAVELLEDDISTANCE\_VALID
  - odometer.h, 28
- odometer.h
  - ODOMETER\_TRAVELLEDDISTANCE\_VALID, 28
- odometer.h, 27
  - EOdometerValidityBits, 27
  - OdometerCallback, 27
  - snsOdometerDeregisterCallback, 28
  - snsOdometerDestroy, 28
  - snsOdometerGetMetaData, 28
  - snsOdometerGetOdometerData, 28
  - snsOdometerInit, 28
  - snsOdometerRegisterCallback, 29
- OdometerCallback
  - odometer.h, 27
- pitchRate
  - TGyroscopeData, 7
- REVERSEGEAR\_REVERSEGEAR\_VALID
  - reverse-gear.h, 30
- rear
  - TSteeringAngleData, 12
- reverse-gear.h
  - REVERSEGEAR\_REVERSEGEAR\_VALID, 30
- reverse-gear.h, 29
  - EReverseGearValidityBits, 30
  - ReverseGearCallback, 29
  - snsReverseGearDeregisterCallback, 30
  - snsReverseGearDestroy, 30
  - snsReverseGearGetMetaData, 30
  - snsReverseGearGetReverseGearData, 30
  - snsReverseGearInit, 31
  - snsReverseGearRegisterCallback, 31
- ReverseGearCallback
  - reverse-gear.h, 29
- rollRate
  - TGyroscopeData, 7
- rotations
  - TWheelspeedAngularDetail, 16
- SENSOR\_CATEGORY\_LOGICAL
  - sns-meta-data.h, 35
- SENSOR\_CATEGORY\_PHYSICAL
  - sns-meta-data.h, 35
- SENSOR\_CATEGORY\_UNKNOWN
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_ACCELERATION
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_GYROSCOPE
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_INCLINATION
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_ODOMETER
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_REVERSE\_GEAR
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_SLIP\_ANGLE
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_STEERING\_ANGLE
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_UNKNOWN
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_VEHICLE\_SPEED
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_VEHICLE\_STATE
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_WHEELSPEED
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_WHEELSPEEDANGULAR
  - sns-meta-data.h, 35
- SENSOR\_TYPE\_WHEELTICK
  - sns-meta-data.h, 35
- SLIPANGLE\_SLIPANGLE\_VALID
  - slip-angle.h, 32
- SNS\_AXLE\_BOTH
  - vehicle-data.h, 39
- SNS\_AXLE\_FRONT
  - vehicle-data.h, 39
- SNS\_AXLE\_REAR
  - vehicle-data.h, 39
- SNS\_AXLE\_UNKNOWN

- vehicle-data.h, 39
- SNS\_BUS
  - vehicle-data.h, 39
- SNS\_CAR
  - vehicle-data.h, 39
- SNS\_MOTORBIKE
  - vehicle-data.h, 39
- SNS\_TRUCK
  - vehicle-data.h, 39
- STEERINGANGLE\_FRONT\_VALID
  - steering-angle.h, 37
- STEERINGANGLE\_REAR\_VALID
  - steering-angle.h, 37
- STEERINGANGLE\_STEERINGWHEEL\_VALID
  - steering-angle.h, 37
- sigmaGyroscope
  - TGyroscopeConfiguration, 6
- sigmaSteeringAngle
  - TSteeringAngleConfiguration, 12
- sigmaSteeringWheelAngle
  - TSteeringAngleConfiguration, 12
- sigmaX
  - TAccelerationConfiguration, 3
- sigmaY
  - TAccelerationConfiguration, 3
- sigmaZ
  - TAccelerationConfiguration, 3
- slip-angle.h
  - SLIPANGLE\_SLIPANGLE\_VALID, 32
- slip-angle.h, 31
  - ESlipAngleValidityBits, 32
  - SlipAngleCallback, 32
  - snsSlipAngleDeregisterCallback, 32
  - snsSlipAngleDestroy, 32
  - snsSlipAngleGetMetaData, 32
  - snsSlipAngleGetSlipAngleData, 33
  - snsSlipAngleInit, 33
  - snsSlipAngleRegisterCallback, 33
- slipAngle
  - TSlipAngleData, 11
- SlipAngleCallback
  - slip-angle.h, 32
- sns-init.h, 33
  - snsDestroy, 34
  - snsGetVersion, 34
  - snsInit, 34
- sns-meta-data.h
  - SENSOR\_CATEGORY\_LOGICAL, 35
  - SENSOR\_CATEGORY\_PHYSICAL, 35
  - SENSOR\_CATEGORY\_UNKNOWN, 35
  - SENSOR\_TYPE\_ACCELERATION, 35
  - SENSOR\_TYPE\_GYROSCOPE, 35
  - SENSOR\_TYPE\_INCLINATION, 35
  - SENSOR\_TYPE\_ODOMETER, 35
  - SENSOR\_TYPE\_REVERSE\_GEAR, 35
  - SENSOR\_TYPE\_SLIP\_ANGLE, 35
  - SENSOR\_TYPE\_STEERING\_ANGLE, 35
  - SENSOR\_TYPE\_UNKNOWN, 35
  - SENSOR\_TYPE\_VEHICLE\_SPEED, 35
  - SENSOR\_TYPE\_VEHICLE\_STATE, 35
  - SENSOR\_TYPE\_WHEELSPEED, 35
  - SENSOR\_TYPE\_WHEELSPEEDANGULAR, 35
  - SENSOR\_TYPE\_WHEELTICK, 35
- sns-meta-data.h, 34
  - ESensorCategory, 35
  - ESensorType, 35
  - getSensorMetaDataList, 35
- snsAccelerationDeregisterCallback
  - acceleration.h, 19
- snsAccelerationDestroy
  - acceleration.h, 20
- snsAccelerationGetAccelerationConfiguration
  - acceleration.h, 20
- snsAccelerationGetAccelerationData
  - acceleration.h, 20
- snsAccelerationGetMetaData
  - acceleration.h, 20
- snsAccelerationInit
  - acceleration.h, 20
- snsAccelerationRegisterCallback
  - acceleration.h, 21
- snsDestroy
  - sns-init.h, 34
- snsGetVersion
  - sns-init.h, 34
- snsGyroscopeDeregisterCallback
  - gyroscope.h, 23
- snsGyroscopeDestroy
  - gyroscope.h, 23
- snsGyroscopeGetConfiguration
  - gyroscope.h, 23
- snsGyroscopeGetGyroscopeData
  - gyroscope.h, 23
- snsGyroscopeGetMetaData
  - gyroscope.h, 24
- snsGyroscopeInit
  - gyroscope.h, 24
- snsGyroscopeRegisterCallback
  - gyroscope.h, 24
- snsInclinationDeregisterCallback
  - inclination.h, 26
- snsInclinationDestroy
  - inclination.h, 26
- snsInclinationGetInclinationData
  - inclination.h, 26
- snsInclinationGetMetaData
  - inclination.h, 26
- snsInclinationInit
  - inclination.h, 26
- snsInclinationRegisterCallback
  - inclination.h, 26
- snsInit
  - sns-init.h, 34
- snsOdometerDeregisterCallback
  - odometer.h, 28
- snsOdometerDestroy



- odometer.h, [28](#)
- snsOdometerGetMetaData
  - odometer.h, [28](#)
- snsOdometerGetOdometerData
  - odometer.h, [28](#)
- snsOdometerInit
  - odometer.h, [28](#)
- snsOdometerRegisterCallback
  - odometer.h, [29](#)
- snsReverseGearDeregisterCallback
  - reverse-gear.h, [30](#)
- snsReverseGearDestroy
  - reverse-gear.h, [30](#)
- snsReverseGearGetMetaData
  - reverse-gear.h, [30](#)
- snsReverseGearGetReverseGearData
  - reverse-gear.h, [30](#)
- snsReverseGearInit
  - reverse-gear.h, [31](#)
- snsReverseGearRegisterCallback
  - reverse-gear.h, [31](#)
- snsSlipAngleDeregisterCallback
  - slip-angle.h, [32](#)
- snsSlipAngleDestroy
  - slip-angle.h, [32](#)
- snsSlipAngleGetMetaData
  - slip-angle.h, [32](#)
- snsSlipAngleGetSlipAngleData
  - slip-angle.h, [33](#)
- snsSlipAngleInit
  - slip-angle.h, [33](#)
- snsSlipAngleRegisterCallback
  - slip-angle.h, [33](#)
- snsSteeringAngleDeregisterCallback
  - steering-angle.h, [37](#)
- snsSteeringAngleDestroy
  - steering-angle.h, [37](#)
- snsSteeringAngleGetConfiguration
  - steering-angle.h, [37](#)
- snsSteeringAngleGetMetaData
  - steering-angle.h, [37](#)
- snsSteeringAngleGetSteeringAngleData
  - steering-angle.h, [38](#)
- snsSteeringAngleInit
  - steering-angle.h, [38](#)
- snsSteeringAngleRegisterCallback
  - steering-angle.h, [38](#)
- snsVehicleSpeedDeregisterCallback
  - vehicle-speed.h, [42](#)
- snsVehicleSpeedDestroy
  - vehicle-speed.h, [43](#)
- snsVehicleSpeedGetMetaData
  - vehicle-speed.h, [43](#)
- snsVehicleSpeedGetVehicleSpeedData
  - vehicle-speed.h, [43](#)
- snsVehicleSpeedInit
  - vehicle-speed.h, [43](#)
- snsVehicleSpeedRegisterCallback
  - vehicle-speed.h, [43](#)
- vehicle-speed.h, [43](#)
- snsVehicleStateDeregisterCallback
  - vehicle-state.h, [45](#)
- snsVehicleStateDestroy
  - vehicle-state.h, [45](#)
- snsVehicleStateGetMetaData
  - vehicle-state.h, [45](#)
- snsVehicleStateGetVehicleStateData
  - vehicle-state.h, [45](#)
- snsVehicleStateInit
  - vehicle-state.h, [46](#)
- snsVehicleStateRegisterCallback
  - vehicle-state.h, [46](#)
- snsWheelGetTireRollingCircumference
  - wheel.h, [48](#)
- snsWheelTickGetMetaData
  - wheel.h, [51](#)
- snsWheelspeedAngularDeregisterCallback
  - wheel.h, [49](#)
- snsWheelspeedAngularDestroy
  - wheel.h, [49](#)
- snsWheelspeedAngularGet
  - wheel.h, [49](#)
- snsWheelspeedAngularGetMetaData
  - wheel.h, [49](#)
- snsWheelspeedAngularInit
  - wheel.h, [49](#)
- snsWheelspeedAngularRegisterCallback
  - wheel.h, [50](#)
- snsWheelspeedDeregisterCallback
  - wheel.h, [50](#)
- snsWheelspeedDestroy
  - wheel.h, [50](#)
- snsWheelspeedGet
  - wheel.h, [50](#)
- snsWheelspeedGetMetaData
  - wheel.h, [50](#)
- snsWheelspeedInit
  - wheel.h, [51](#)
- snsWheelspeedRegisterCallback
  - wheel.h, [51](#)
- snsWheeltickDeregisterCallback
  - wheel.h, [51](#)
- snsWheeltickDestroy
  - wheel.h, [51](#)
- snsWheeltickGetWheelticks
  - wheel.h, [52](#)
- snsWheeltickGetWheelticksCountMax
  - wheel.h, [52](#)
- snsWheeltickGetWheelticksPerRevolution
  - wheel.h, [52](#)
- snsWheeltickInit
  - wheel.h, [52](#)
- snsWheeltickRegisterCallback
  - wheel.h, [52](#)
- speedOfWheel
  - TWheelspeedDetail, [16](#)
- status

- TInclinationData, 8
- steering-angle.h
  - STEERINGANGLE\_FRONT\_VALID, 37
  - STEERINGANGLE\_REAR\_VALID, 37
  - STEERINGANGLE\_STEERINGWHEEL\_VALID, 37
- steering-angle.h, 36
  - ESteeringAngleValidityBits, 37
  - snsSteeringAngleDeregisterCallback, 37
  - snsSteeringAngleDestroy, 37
  - snsSteeringAngleGetConfiguration, 37
  - snsSteeringAngleGetMetaData, 37
  - snsSteeringAngleGetSteeringAngleData, 38
  - snsSteeringAngleInit, 38
  - snsSteeringAngleRegisterCallback, 38
  - SteeringAngleCallback, 36
- SteeringAngleCallback
  - steering-angle.h, 36
- steeringRatio
  - TSteeringAngleConfiguration, 12
- steeringWheel
  - TSteeringAngleData, 12
- TAccelerationConfiguration, 2
  - 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, 6
  - typeBits, 6
  - validityBits, 6
- TGyroscopeData, 7
  - pitchRate, 7
  - rollRate, 7
  - temperature, 7
  - timestamp, 7
  - validityBits, 7
  - yawRate, 8
- TInclinationData, 8
  - longitudinalGradientRoadway, 8
  - status, 8
  - timestamp, 8
  - traverseGradientRoadway, 8
  - validityBits, 8
- TOdometerData, 9
  - timestamp, 9
  - travelledDistance, 9
  - validityBits, 9
- TReverseGearData, 9
  - isReverseGear, 10
  - timestamp, 10
  - validityBits, 10
- TSensorMetaData, 10
  - category, 10
  - cycleTime, 10
  - type, 10
  - version, 10
- TSlipAngleData, 11
  - slipAngle, 11
  - timestamp, 11
  - validityBits, 11
- TSteeringAngleConfiguration, 11
  - sigmaSteeringAngle, 12
  - sigmaSteeringWheelAngle, 12
  - steeringRatio, 12
- TSteeringAngleData, 12
  - front, 12
  - rear, 12
  - steeringWheel, 12
  - timestamp, 12
  - validityBits, 12
- TVehicleSpeedData, 13
  - timestamp, 13
  - validityBits, 13
  - vehicleSpeed, 13
- TVehicleStateData, 13
  - antiLockBrakeSystemActive, 14
  - brakeActive, 14
  - electronicStabilityProgramActive, 14
  - timestamp, 14
  - tractionControlActive, 14
  - validityBits, 14
- TWheelspeed, 14
  - elements, 14
  - timestamp, 14
- TWheelspeedAngular, 15
  - elements, 15
  - timestamp, 15
- TWheelspeedAngularDetail, 15
  - id, 16
  - rotations, 16
- TWheelspeedDetail, 16
  - id, 16
  - speedOfWheel, 16

- TWheeltickDetail, 16
  - wheeltickCounter, 17
  - wheeltickIdentifier, 17
- TWheelticks, 17
  - elements, 17
  - timestamp, 17
- temperature
  - TAccelerationData, 4
  - TGyroscopeData, 7
- timestamp
  - TAccelerationData, 4
  - TGyroscopeData, 7
  - TInclinationData, 8
  - TOdometerData, 9
  - TReverseGearData, 10
  - TSlipAngleData, 11
  - TSteeringAngleData, 12
  - TVehicleSpeedData, 13
  - TVehicleStateData, 14
  - TWheelspeed, 14
  - TWheelspeedAngular, 15
  - TWheelticks, 17
- tractionControlActive
  - TVehicleStateData, 14
- travelledDistance
  - TOdometerData, 9
- traverseGradientRoadway
  - TInclinationData, 8
- type
  - TSensorMetaData, 10
- typeBits
  - TAccelerationConfiguration, 3
  - TGyroscopeConfiguration, 6
- VEHICLESPEED\_VEHICLESPEED\_VALID
  - vehicle-speed.h, 42
- VEHICLESTATE\_ANTILOCKBRAKESYSTEMACTIVE\_VALID
  - vehicle-state.h, 45
- VEHICLESTATE\_BRAKEACTIVE\_VALID
  - vehicle-state.h, 45
- VEHICLESTATE\_ELECTRONICSTABILITYPROGRAMACTIVE\_VALID
  - vehicle-state.h, 45
- VEHICLESTATE\_TRACTIONCONTROLACTIVE\_VALID
  - vehicle-state.h, 45
- validityBits
  - TAccelerationConfiguration, 3
  - TAccelerationData, 4
  - TGyroscopeConfiguration, 6
  - TGyroscopeData, 7
  - TInclinationData, 8
  - TOdometerData, 9
  - TReverseGearData, 10
  - TSlipAngleData, 11
  - TSteeringAngleData, 12
  - TVehicleSpeedData, 13
  - TVehicleStateData, 14
- vehicle-data.h
  - SNS\_AXLE\_BOTH, 39
  - SNS\_AXLE\_FRONT, 39
  - SNS\_AXLE\_REAR, 39
  - SNS\_AXLE\_UNKNOWN, 39
  - SNS\_BUS, 39
  - SNS\_CAR, 39
  - SNS\_MOTORBIKE, 39
  - SNS\_TRUCK, 39
- vehicle-data.h, 38
  - EAxleType, 39
  - EVehicleType, 39
  - vehicleDataGetDistCoG2RefPoint, 39
  - vehicleDataGetDistFrontAxle2RefPoint, 39
  - vehicleDataGetDistRearAxle2RefPoint, 40
  - vehicleDataGetDrivenAxles, 40
  - vehicleDataGetFrontAxleTrackWidth, 40
  - vehicleDataGetSeatCount, 40
  - vehicleDataGetTrackWidth, 40
  - vehicleDataGetVehicleMass, 41
  - vehicleDataGetVehicleType, 41
  - vehicleDataGetVehicleWidth, 41
  - vehicleDataGetWheelBase, 41
- vehicle-speed.h
  - VEHICLESPEED\_VEHICLESPEED\_VALID, 42
- vehicle-speed.h, 41
  - EVehicleSpeedValidityBits, 42
  - snsVehicleSpeedDeregisterCallback, 42
  - snsVehicleSpeedDestroy, 43
  - snsVehicleSpeedGetMetaData, 43
  - snsVehicleSpeedGetVehicleSpeedData, 43
  - snsVehicleSpeedInit, 43
  - snsVehicleSpeedRegisterCallback, 43
  - VehicleSpeedCallback, 42
- vehicle-state.h
  - VEHICLESTATE\_ANTILOCKBRAKESYSTEMACTIVE\_VALID, 45
  - VEHICLESTATE\_BRAKEACTIVE\_VALID, 45
  - VEHICLESTATE\_ELECTRONICSTABILITYPROGRAMACTIVE\_VALID, 45
  - VEHICLESTATE\_TRACTIONCONTROLACTIVE\_VALID, 45
- vehicle-state.h, 44
  - EVehicleStateValidityBits, 45
  - snsVehicleStateDeregisterCallback, 45
  - snsVehicleStateDestroy, 45
  - snsVehicleStateGetMetaData, 45
  - snsVehicleStateGetVehicleStateData, 45
  - snsVehicleStateInit, 46
  - snsVehicleStateRegisterCallback, 46
  - VehicleStateCallback, 44
- vehicleDataGetDistCoG2RefPoint
  - vehicle-data.h, 39
- vehicleDataGetDistFrontAxle2RefPoint
  - vehicle-data.h, 39
- vehicleDataGetDistRearAxle2RefPoint
  - vehicle-data.h, 40
- vehicleDataGetDrivenAxles

- vehicle-data.h, 40
- vehicleDataGetFrontAxleTrackWidth
  - vehicle-data.h, 40
- vehicleDataGetSeatCount
  - vehicle-data.h, 40
- vehicleDataGetTrackWidth
  - vehicle-data.h, 40
- vehicleDataGetVehicleMass
  - vehicle-data.h, 41
- vehicleDataGetVehicleType
  - vehicle-data.h, 41
- vehicleDataGetVehicleWidth
  - vehicle-data.h, 41
- vehicleDataGetWheelBase
  - vehicle-data.h, 41
- vehicleSpeed
  - TVehicleSpeedData, 13
- VehicleSpeedCallback
  - vehicle-speed.h, 42
- VehicleStateCallback
  - vehicle-state.h, 44
- version
  - TSensorMetaData, 10
- WHEEL\_AXLE\_FRONT
  - wheel.h, 48
- WHEEL\_AXLE\_NONDRIVEN
  - wheel.h, 48
- WHEEL\_AXLE\_REAR
  - wheel.h, 48
- WHEEL\_INVALID
  - wheel.h, 48
- WHEEL\_LEFT\_FRONT
  - wheel.h, 48
- WHEEL\_LEFT\_REAR
  - wheel.h, 48
- WHEEL\_RIGHT\_FRONT
  - wheel.h, 48
- WHEEL\_RIGHT\_REAR
  - wheel.h, 48
- WHEEL\_UNKNOWN
  - wheel.h, 48
- WHEEL\_NUM\_ELEMENTS
  - wheel.h, 47
- wheel.h
  - WHEEL\_AXLE\_FRONT, 48
  - WHEEL\_AXLE\_NONDRIVEN, 48
  - WHEEL\_AXLE\_REAR, 48
  - WHEEL\_INVALID, 48
  - WHEEL\_LEFT\_FRONT, 48
  - WHEEL\_LEFT\_REAR, 48
  - WHEEL\_RIGHT\_FRONT, 48
  - WHEEL\_RIGHT\_REAR, 48
  - WHEEL\_UNKNOWN, 48
- wheel.h, 46
  - EWheelId, 48
  - snsWheelGetTireRollingCircumference, 48
  - snsWheelTickGetMetaData, 51
  - snsWheelspeedAngularDeregisterCallback, 49
  - snsWheelspeedAngularDestroy, 49
  - snsWheelspeedAngularGet, 49
  - snsWheelspeedAngularGetMetaData, 49
  - snsWheelspeedAngularInit, 49
  - snsWheelspeedAngularRegisterCallback, 50
  - snsWheelspeedDeregisterCallback, 50
  - snsWheelspeedDestroy, 50
  - snsWheelspeedGet, 50
  - snsWheelspeedGetMetaData, 50
  - snsWheelspeedInit, 51
  - snsWheelspeedRegisterCallback, 51
  - snsWheeltickDeregisterCallback, 51
  - snsWheeltickDestroy, 51
  - snsWheeltickGetWheelticks, 52
  - snsWheeltickGetWheelticksCountMax, 52
  - snsWheeltickGetWheelticksPerRevolution, 52
  - snsWheeltickInit, 52
  - snsWheeltickRegisterCallback, 52
  - WHEEL\_NUM\_ELEMENTS, 47
  - WheelspeedAngularCallback, 47
  - WheelspeedCallback, 48
  - WheeltickCallback, 48
- WheelspeedAngularCallback
  - wheel.h, 47
- WheelspeedCallback
  - wheel.h, 48
- WheeltickCallback
  - wheel.h, 48
- wheeltickCounter
  - TWheeltickDetail, 17
- wheeltickIdentifier
  - TWheeltickDetail, 17
- x
  - TAccelerationData, 4
  - TDistance3D, 5
- y
  - TAccelerationData, 4
  - TDistance3D, 5
- yawRate
  - TGyroscopeData, 8
- z
  - TAccelerationData, 4
  - TDistance3D, 5