GENIVI SensorsService

Component Specification

Release 5.0.0
Status: Released
26.01.2017

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

Abstract:
This document describes the API 5.0.0 of the **SensorsService** Abstract Component.

Keywords:
SensorsService, Sensors, Positioning API.

Table of Contents

Change History

Version	Date	Author	Change
0.1	27.08.2013	M. Residori	Document Created
0.2	18.11.2013	M. Residori	Document generated from the Enterprise Architect Model
0.3	27.03.2014	M. Residori	Added copyright notes
3.0.0-alpha	24.04.2014	M. Residori	Changed license version from 3.0 to 4.0
3.0.0-alpha	10.12.2014	M. Residori	Updated API description
3.0.0-alpha	20.01.2015	H. Schmidt	Fix copy/paste error
3.0.0	20.01.2015	M. Residori	Changed status to “Released” (after System Architecture Team approval)
3.0.1	01.04.2015	H. Schmidt	Bugfix in gyroscope/acceleration API
4.0.0.-alpha	16.12.2015	M. Residori	Updated API description
4.0.0	25.01.2016	M. Residori	Release 4.0.0
5.0.0	26.01.2017	H.Schmidt M. Residori	Release 5.0.0

1. Introduction

This document describes the API of the SensorsService component.

2. Terminology

Term	***Description***
GNSS	Global Navigation Satellite System

3. Requirements

1. Requirements Diagram

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

The requirements are organized in four groups:
1. SW-POS: general requirements
2. SW-GNSS: requirements related to the GNSS receiver
3. SW-SNS: requirements related to the vehicle sensors
4. SW-ENP: requirements related to enhanced positioning


req Requirements
SW-POS
Sensor Meta-Data
tags
Originator = Thomas Himbacher (BMW)
Priority = P1
Rationale =
Sensor Signal Timestamp
tags
Originator = Thomas Bader (BMW)
Priority = P1
Rationale =
Signal Measurement Units
tags
Originator = Thomas Bader (BMW)
Priority = P1
Rationale =
Car Configuration Data
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Signal Values Type Compatibility
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Access to Sensor Services
tags
Originator = Thomas Himbacher (BMW)
Priority = P2
Rationale =
Sensor Directory
tags
Originator = Thomas Himbacher (BMW)
Priority = P2
Rationale =
Support of different Global Navigation Satellite Systems (GNSS) to calculate the current position.
tags
Originator = Philippe Colliot (PSA)
Priority = P2
Rationale = <memo>
SW-POS-GNSS
GNSS Service
tags
Originator = Thomas Himbacher (BMW)
Priority = P1
Rationale =
Extended GNSS Service
tags
Originator = Thomas Himbacher (BMW)
Priority = P2
Rationale =
PPS Signal
tags
Originator = Carsten Isert (BMW)
Priority = P2
Rationale =
AGPS Support
tags
Originator = Thomas Bader (BMW)
Priority = P3
Rationale =
SW-POS-SNS
VehicleSpeed Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Odometer Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Wheel Tick/Speed Sensor Service
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Simple Gyroscope Sensor Service
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Extended Gyroscope Sensor Service
tags
Originator = Thomas Bader (BMW)
Priority = P3
Rationale =
ReverseGear Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Accelerator Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Extended Acceleration Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P3
Rationale =
Inclination Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P3
Rationale =
Slip Angle Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P3
Rationale =
Vehicle State Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P3
Rationale =
SteeringAngle Sensor
tags
Originator = Thomas Bader (BMW)
Priority = P3
Rationale =
SW-POS-ENP
Enhanced Position
tags
Originator = Thomas Himbacher (BMW)
Priority = P1
Rationale =
Provides Data via IPC
tags
Originator = Thomas Bader (BMW)
Priority = P1
Rationale =
Forward a Set of Sensor Values
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =
Data Latency for GNSS and DR Signals
tags
Originator = Thomas Bader (BMW)
Priority = P2
Rationale =

Figure: 1

AGPS Support		
«GFunctionalRequirement»	*Priority:* Medium	

Description:
The software platform provides the possibility to inject AGPS "Assisted GPS" data to the GPS device.

Rationale:
This allows to speed up the time to get a valid (fixed) GPS position.

Forward a Set of Sensor Values		
«GFunctionalRequirement»	*Priority:* Medium	

Description:
The Enhanced Position contains in addition to the Position and Course values as well a set of sensor data.
- yawRate in degrees per second
- filter status
- accuracy information in form of sigma values for every direction [m] and the covariance between latitude and longitude in m^2.
- number of used, tracked and visible satellites.

Rational:
Some clients (e.g. Map Matcher) needs the basic DR filtered position specific sensor values as additional input for the decision algorithm.

Provides Data via IPC		
«GFunctionalRequirement»	*Priority:* Medium	

Description:
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.

Rational:
Several SW components in the system are clients for the result of the filtered position and need to access the data.

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

The interfaces are defined in such a way that client applications don´t need to know the details of the GNSS in use (e.g. GPS, Galileo, GLONASS, Compass).

Accelerator Sensor		
«GFunctionalRequirement»	*Priority:* Medium	

Description:
The software platform provides a sensor, which delivers the vehicle acceleration in the driving direction (x Axis, see reference system). The sensor value is delivered in m/s^2. 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	

Description:
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:
- Position expressed as WGS 84 longitude and latitude (unit is tenth of microdegree (degree x 10^-7^))
- the Altitude 'above mean sea level' in meters (corrected by GeoID)
- Heading in degrees relative to the true north
- Climb
- Speed in meters per seconds, positive in the forward direction

Rational:
Other SW-components on the same platform want to access the improved GNSS position, which is calculated by a dead reckoning algorithm.

Extended Acceleration Sensor

«GFunctionalRequirement»	*Priority:* Low	

Description:
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.

Rational:
Used for optimizing the dead reckoning solution.

Extended GNSS Service

«GFunctionalRequirement»	*Priority:* Medium	

Description:
The software platform provides an extension to the GNSS Service with optional information.

Accuracy:
- fixStatus
- hdop, pdop, vdop
- numberOfSatellites
- sigmaLatitude, sigmaLongitude, sigmaAltitude

Satellite Details:
- Information per satellite: azimuth, elevation, inUse, SatelliteId, signalNoiseRatio

Course Details:
- speed for 3-axis

Antenna:
- Antenna Position in 3D coordinates in relation to the reference point (see reference system).

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,

GPS, etc)

Rational:
These data are used for improved positioning based on GNSS.

Extended Gyroscope Sensor Service

«GFunctionalRequirement»	*Priority:* Low	

Description:
The software platform includes the sensor that delivers
- pitch rate
- roll rate

This sensor values extend the simple gyroscope sensor.
Sign of is defined by rule of right hand (thumb direction: left and front, see reference system).
Car configuration data need to provide position angles according to vehicle reference system.

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

GNSS Service

«GFunctionalRequirement»	*Priority:* High	

Description:
The software platform includes a service that provides the following GNSS Signals updated at least with 1Hz frequency:
Position:
- position expressed as WGS 84 altitude, longitude and latitude in tenth of microdegree (degree x 10^-7^)

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

Timestamp and date as UTC.

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

PPS Signal

«GFunctionalRequirement»	*Priority:* Medium	

Description:
1) For accurate timing the 1 PPS (pulse per second) signal from the GPS receiver is provided within the positioning framework.
The PPS is a hardware signal which is a UTC synchronized pulse.
The duration between the pulses is 1s +/- 40ns and the duration of the pulse isconfigurable (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 checks 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	

Description:
The software platform provides the following information about the Sensor and the related output Signals:
- Sensor Identifier that is unique within the system
- Sensor Category (Physical/Logical)
- Sensor Type (GPS, Odometer, Map Matching, etc.)
- Sensor Sub-Type (ordinary GPS, differential GPS, etc.)
- Output Signals (Longitude, Latitude, Course, Speed, etc.)
- Output Signal Sampling Frequency (1 Hz, 10 Hz, irregular, etc.)
- Output Signal Measurement Units (kilometers per hour; meters per second; etc.)

Rational:
Sensor clients need that information in order to correctly handle data provided by sensor service and to adapt to the variation in the signal data delivery.

Sensor Signal Timestamp

«GFunctionalRequirement»	*Priority:* High	

Description:
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.

Rational:
Measurement timestamps are important for proper functioning of most processing algorithms. For instance, algorithms for sensor calibration and dead reckoning typically use data from multiple sensors in conjunction, e.g. logical sensor.

Signal Measurement Units

«GFunctionalRequirement»	*Priority:* High	

Description:
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.

Rational:
This decouples the client applications from the implementation details of individual sensor devices.

Signal Values Type Compatibility

«GFunctionalRequirement»	*Priority:* Medium	

Description:
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.

Rational:
Sensor service clients are able to use multiple Sensor Services without changes in the interfaces.

Simple Gyroscope Sensor Service

«GFunctionalRequirement»	*Priority:* Medium	

Description:
The software platform includes the Sensor that delivers
- yaw rate: the rate of the vehicle heading change
-temperature
- status:(temperature compensated or not, etc)
at the frequency of at least 5Hz. Unit of yaw rate is "degrees per second".

Sign of yaw rate is defined by rule of right hand (thumb direction: up)
(see reference system)

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

Slip Angle Sensor

«GFunctionalRequirement»	*Priority:* Low	

Description:
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).

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

SteeringAngle Sensor

«GFunctionalRequirement»	*Priority:* Low	

Description:
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.

Rational:
Is used as additional element for plausibilisation of the yaw rate in the dead reckoning module.

Vehicle State Sensor		
«GFunctionalRequirement»	*Priority:* Low	

Description:
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

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

VehicleSpeed Sensor		
«GFunctionalRequirement»	*Priority:* Medium	

Description:
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.

Rational:
Vehicle speed is sometimes the only speed related signal available to the head unit.

Wheel Tick/Speed Sensor Service		
«GFunctionalRequirement»	*Priority:* Medium	

Description:
The software platform provides a Sensor that delivers the running counter of partial wheel revolutions at the frequency of at least 5Hz or the already calculated wheelspeed (speed in [m/s] or angular speed).

The resolution of a single wheel revolution (i.e. the number of ticks per revolution) is included with the Sensor Service meta-data.

This identifiers specify the wheel of measurement:
0: Average of non driven axle
1: Left front wheel
2: Right front wheel
3: Left rear wheel
4: Right rear wheel
Unit: [ticks].

Rational:
This Sensor typically registers ‘ticks’ from a wheel, adds them up and sends to the vehicle bus with a certain interval. The number of ‘ticks’ per complete wheel revolution is known in advance. In some cases, the data from multiple wheels are averaged. Other implementations send the already precalculated speed per wheel or axle, which is a valid replacement for most use cases.

4. Architecture

1. *SensorsService*

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

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

2. *SensorsService Diagram*

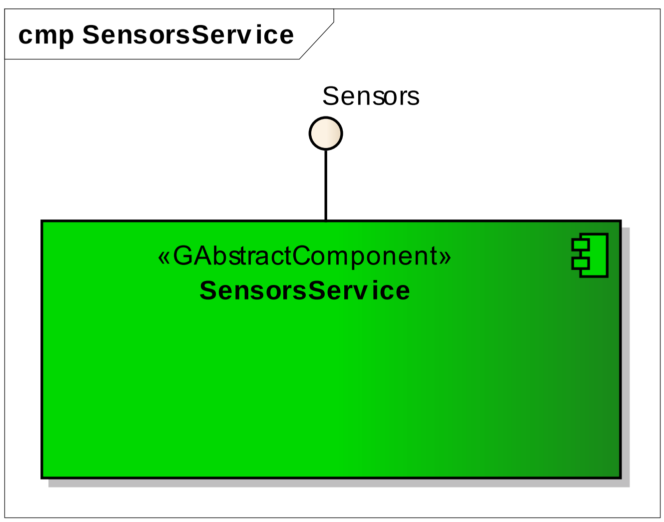

Figure: 2

3. *Traceability Diagram*

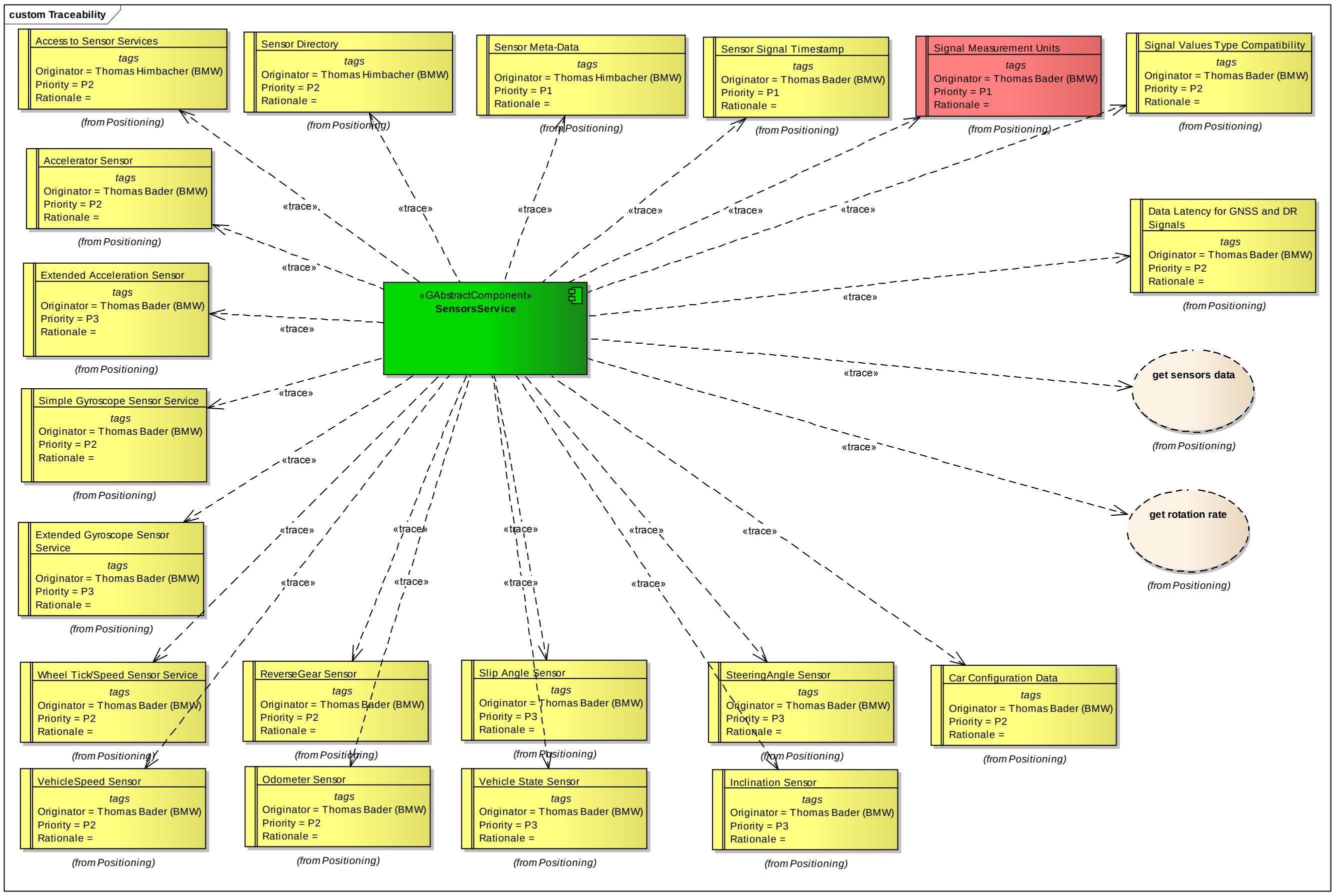

Figure: 3

1. *Context*

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

2. Context Diagram

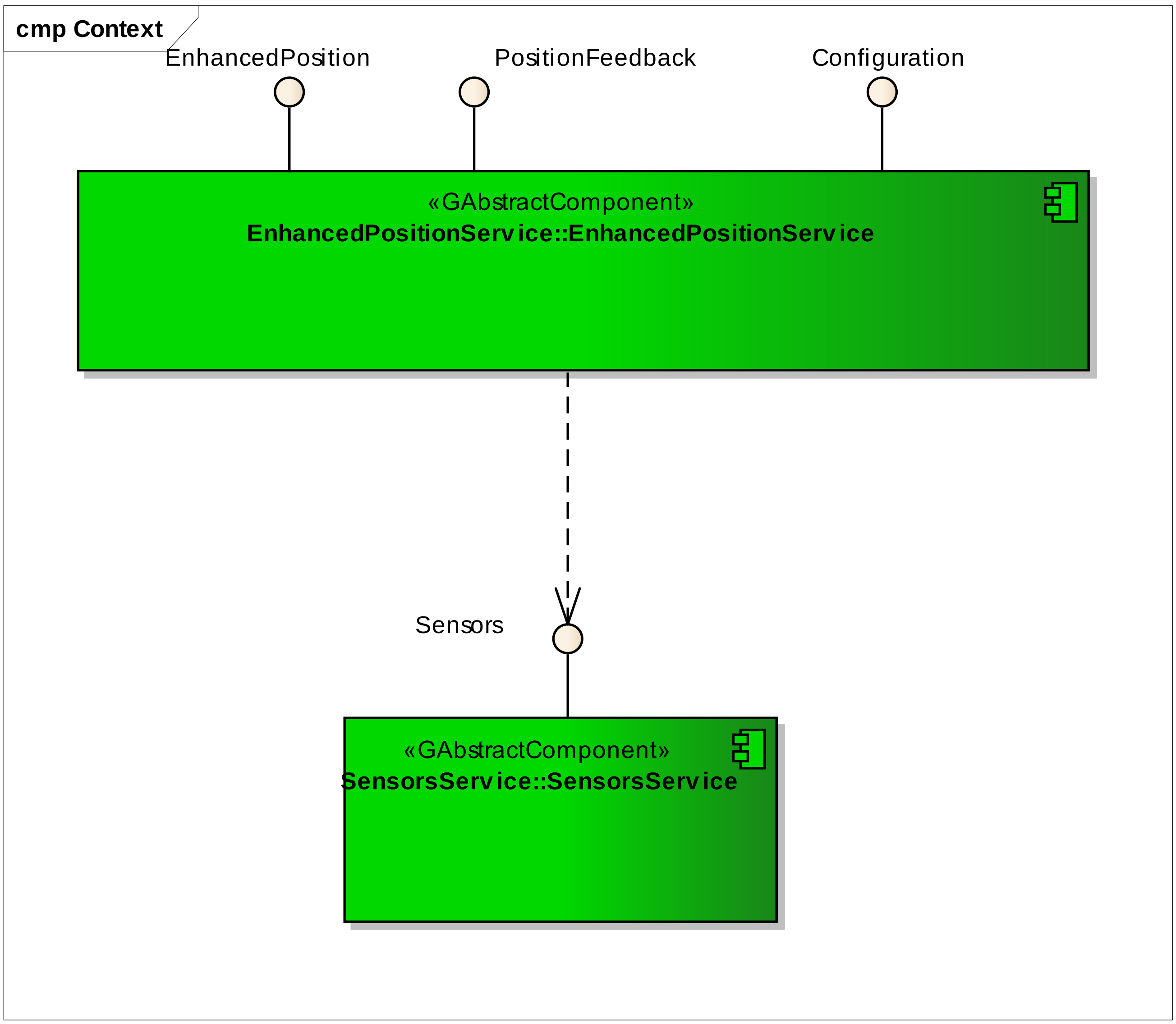

Figure: 4

SensorsService

Generated by Doxygen 1.8.9.1

Wed Feb 22 2017 07:30:55

Contents

1 Class Documentation

1.1 TAccelerationConfiguration Struct Reference

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

Public Attributes

- float dist2RefPointX
- float dist2RefPointY
- float dist2RefPointZ
- float angleYaw
- float anglePitch
- float angleRoll
- float sigmaX
- float sigmaY
- float sigmaZ
- uint32_t typeBits
- uint32_t validityBits

1.1.1 Detailed Description

Static configuration data for the acceleration sensor service.

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

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

Notes

- the angles are frequently called "Euler angles" and the rotations "Euler rotations"
- a different order of the rotations would lead to a different orientation
- as the vehicle axis system is right-handed, also the accelerometer axis system must be right-handed

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

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

When the accelerometer axes are not aligned with the vehicle axes, i.e. if any of the angles (angleYaw, anglePitch, angleRoll) is not zero then the raw measurement values of the accelerometer X, Y, Z axes may have to be be transformed to the vehicle axis system by the client of this interface, depending on the type of application. Raw measurements are provided in TAccelerationData 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^2].

1.1.2.8 float TAccelerationConfiguration::sigmaY

Standard error estimate of the y-acceleration [m/s^2].

1.1.2.9 float TAccelerationConfiguration::sigmaZ

Standard error estimate of the z-acceleration [m/s^2].

1.1.2.10 uint32_t TAccelerationConfiguration::typeBits

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

1.1.2.11 uint32_t TAccelerationConfiguration::validityBits

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

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

- acceleration.h

1.2 TAccelerationData Struct Reference

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

Public Attributes

- uint64_t timestamp
- float x
- float y
- float z
- float temperature
- uint32_t measurementInterval
- 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 uint32_t TAccelerationData::measurementInterval

Measurement interval over which the accelerometer signal has been acquired. Unit: micro-seconds [us]. This may slightly differ from the timestamp difference, e.g. in case of transmission jitter before timestamping. Providing the measurement interval allows thus

- a more accurate integration of accelerometer measurements.
- correct usage of the first sample
- adding consistency checks

1.2.2.2 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.

1.2.2.3 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.

1.2.2.4 uint32_t TAccelerationData::validityBits

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

1.2.2.5 float TAccelerationData::x

The acceleration in direction of the X-axis of the accelerometer sensor [m/s^2].

1.2.2.6 float TAccelerationData::y

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

1.2.2.7 float TAccelerationData::z

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

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

- acceleration.h

1.3 TDistance3D Struct Reference

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

Public Attributes

- float x
- float y
- float z

1.3.1 Detailed Description

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

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

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

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

1.3.2 Member Data Documentation

1.3.2.1 float TDistance3D::x

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

1.3.2.2 float TDistance3D::y

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

1.3.2.3 float TDistance3D::z

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

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

- vehicle-data.h

1.4 TGyroscopeConfiguration Struct Reference

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

Public Attributes

- float angleYaw
- float anglePitch
- float angleRoll
- float momentOfYawInertia
- float sigmaGyroscope
- uint32_t typeBits
- uint32_t validityBits

1.4.1 Detailed Description

Static configuration data for the gyroscope sensor service.

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

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

Notes

- the angles are frequently called "Euler angles" and the rotations "Euler rotations"
- a different order of the rotations would lead to a different orientation
- as the vehicle axis system is right-handed, also the gyroscope axis system must be right-handed

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

- Xv is in the horizontal plane, pointing forwards
- Yv is in the horizontal plane, pointing to the left
- Zv is perpendicular to the horizontal plane, pointing upwards For an illustration, see https://collab.genivi.org/wiki/display/genivi/LBSSensorServiceRequirementsBorg#LBSSensorServiceRequirementsBorg-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 be transformed to the vehicle axis system by the client of this interface, depending on the type of application. Raw measurements are provided in TGyroscopeData 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 [kg*m^2]. The pitch and roll inertia moments are not provided as they are not relevant for positioning.

1.4.2.5 float TGyroscopeConfiguration::sigmaGyroscope

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

1.4.2.6 uint32_t TGyroscopeConfiguration::typeBits

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

1.4.2.7 uint32_t TGyroscopeConfiguration::validityBits

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

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

- gyroscope.h

1.5 TGyroscopeData Struct Reference

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

Public Attributes

- uint64_t timestamp
- float yawRate
- float pitchRate
- float rollRate
- float temperature
- uint32_t measurementInterval
- 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 uint32_t TGyroscopeData::measurementInterval

Measurement interval over which the gyroscope signal has been acquired. Unit: micro-seconds [us]. This may slightly differ from the timestamp difference, e.g. in case of transmission jitter before timestamping. Providing the measurement interval allows thus

- a more accurate integration of gyroscope measurements.
- correct usage of the first sample
- adding consistency checks

1.5.2.2 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, *if* gyroscope axes are aligned with vehicle axes i.e. if all euler angles are zero in TGyroscopeConfiguration.

1.5.2.3 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, *if* gyroscope axes are aligned with vehicle axes i.e. if all euler angles are zero in TGyroscopeConfiguration.

1.5.2.4 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.

1.5.2.5 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.

1.5.2.6 uint32_t TGyroscopeData::validityBits

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

1.5.2.7 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. Preferrably 50Hz. A rotation to the left is indicated by a positive sign, *if* gyroscope axes are aligned with vehicle axes i.e. if all euler angles are zero in TGyroscopeConfiguration.

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 TSensorStatus Struct Reference

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

Public Attributes

- uint64_t timestamp

- ESensorStatus status
- uint32_t validityBits

1.10.1 Detailed Description

Container for sensor status information

1.10.2 Member Data Documentation

1.10.2.1 ESensorStatus TSensorStatus::status

Status of the sensor

1.10.2.2 uint64_t TSensorStatus::timestamp

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

1.10.2.3 uint32_t TSensorStatus::validityBits

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

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

- sns-status.h

1.11 TSlipAngleData Struct Reference

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

Public Attributes

- uint64_t timestamp
- float slipAngle
- uint32_t validityBits

1.11.1 Detailed Description

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

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

1.11.2 Member Data Documentation

1.11.2.1 float TSlipAngleData::slipAngle

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

1.11.2.2 uint64_t TSlipAngleData::timestamp

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

1.11.2.3 uint32_t TSlipAngleData::validityBits

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

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

- slip-angle.h

1.12 TSteeringAngleConfiguration Struct Reference

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

Public Attributes

- float sigmaSteeringAngle
- float sigmaSteeringWheelAngle
- float steeringRatio

1.12.1 Detailed Description

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

1.12.2 Member Data Documentation

1.12.2.1 float TSteeringAngleConfiguration::sigmaSteeringAngle

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

1.12.2.2 float TSteeringAngleConfiguration::sigmaSteeringWheelAngle

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

1.12.2.3 float TSteeringAngleConfiguration::steeringRatio

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

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

- steering-angle.h

1.13 TSteeringAngleData Struct Reference

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

Public Attributes

- uint64_t timestamp
- float front
- float rear
- float steeringWheel
- uint32_t validityBits

1.13.1 Detailed Description

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

You must check the valid bits before usage.

1.13.2 Member Data Documentation

1.13.2.1 float TSteeringAngleData::front

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

1.13.2.2 float TSteeringAngleData::rear

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

1.13.2.3 float TSteeringAngleData::steeringWheel

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

1.13.2.4 uint64_t TSteeringAngleData::timestamp

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

1.13.2.5 uint32_t TSteeringAngleData::validityBits

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

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

- steering-angle.h

1.14 TVehicleDataConfiguration Struct Reference

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

Public Attributes

- EVehicleType vehicleType
- EAxleType drivenAxles
- uint8_t seatCount
- float trackWidth
- float frontAxleTrackWidth
- float wheelBase
- float vehicleMass
- float vehicleWidth
- TDistance3D distCoG2RefPoint
- TDistance3D distFrontAxle2RefPoint
- TDistance3D distRearAxle2RefPoint
- uint32_t validityBits

1.14.1 Detailed Description

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

1.14.2 Member Data Documentation

1.14.2.1 TDistance3D TVehicleDataConfiguration::distCoG2RefPoint

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

1.14.2.2 TDistance3D TVehicleDataConfiguration::distFrontAxle2RefPoint

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

1.14.2.3 TDistance3D TVehicleDataConfiguration::distRearAxle2RefPoint

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

1.14.2.4 EAxleType TVehicleDataConfiguration::drivenAxles

Type of the driven axles of the vehicle.

1.14.2.5 float TVehicleDataConfiguration::frontAxleTrackWidth

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

1.14.2.6 uint8_t TVehicleDataConfiguration::seatCount

Number of seats of the vehicle.

1.14.2.7 float TVehicleDataConfiguration::trackWidth

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

1.14.2.8 uint32_t TVehicleDataConfiguration::validityBits

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

1.14.2.9 float TVehicleDataConfiguration::vehicleMass

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

1.14.2.10 EVehicleType TVehicleDataConfiguration::vehicleType

Type of the vehicle.

1.14.2.11 float TVehicleDataConfiguration::vehicleWidth

Width of the vehicle. Unit: [m].

1.14.2.12 float TVehicleDataConfiguration::wheelBase

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

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

- vehicle-data.h

1.15 TVehicleSpeedData Struct Reference

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

Public Attributes

- uint64_t timestamp
- float vehicleSpeed
- uint32_t measurementInterval
- uint32_t validityBits

1.15.1 Detailed Description

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

1.15.2 Member Data Documentation

1.15.2.1 uint32_t TVehicleSpeedData::measurementInterval

Measurement interval over which the vehicle speed signal has been acquired. Unit: micro-seconds [us]. This may slightly differ from the timestamp difference, e.g. in case of transmission jitter before timestamping. Providing the measurement interval allows thus

- a more accurate integration of vehicle speed measurements.
- correct usage of the first sample
- adding consistency checks

1.15.2.2 uint64_t TVehicleSpeedData::timestamp

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

1.15.2.3 uint32_t TVehicleSpeedData::validityBits

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

1.15.2.4 float TVehicleSpeedData::vehicleSpeed

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

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

- vehicle-speed.h

1.16 TVehicleStateData Struct Reference

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

Public Attributes

- uint64_t timestamp
- bool antiLockBrakeSystemActive
- bool brakeActive
- bool electronicStabilityProgramActive
- bool tractionControlActive
- uint32_t validityBits

1.16.1 Detailed Description

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

1.16.2 Member Data Documentation

1.16.2.1 bool TVehicleStateData::antiLockBrakeSystemActive

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

1.16.2.2 bool TVehicleStateData::brakeActive

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

1.16.2.3 bool TVehicleStateData::electronicStabilityProgramActive

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

1.16.2.4 uint64_t TVehicleStateData::timestamp

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

1.16.2.5 bool TVehicleStateData::tractionControlActive

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

1.16.2.6 uint32_t TVehicleStateData::validityBits

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

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

- vehicle-state.h

1.17 TWheelConfiguration Struct Reference

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

Public Attributes

- EWheelUnit wheelUnit
- uint8_t axleIndex
- uint8_t wheelIndex
- uint16_t wheelticksPerRevolution
- float tireRollingCircumference
- float dist2RefPointX
- float dist2RefPointY
- float dist2RefPointZ
- uint32_t statusBits
- uint32_t validityBits

1.17.1 Detailed Description

Configuration data for an individual wheel. Most configuration parameters are static for a given wheel. The status bits WHEEL_CONFIG_DRIVEN, WHEEL_CONFIG_STEERED, WHEEL_CONFIG_DIFF_LOCK are considered as dynamic configuration data. I.e. those status bits may change dynamically during runtime.

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

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

1.17.2 Member Data Documentation

1.17.2.1 uint8_t TWheelConfiguration::axleIndex

[Static] Identification of the axle on which the wheel is mounted Axles are numbered consecutively from front to rear. 0: unknown/unspecified 1: front axle 2: 2nd axle (rear axle on a typical 2-axle vehicle) 3: 3rd axle ...

1.17.2.2 float TWheelConfiguration::dist2RefPointX

[Static] Distance of the wheel center from the vehicle reference point (x-coordinate) [m].

1.17.2.3 float TWheelConfiguration::dist2RefPointY

[Static] Distance of the wheel center from the vehicle reference point (y-coordinate) [m].

1.17.2.4 float TWheelConfiguration::dist2RefPointZ

[Static] Distance of the wheel center from the vehicle reference point (z-coordinate) [m].

1.17.2.5 uint32_t TWheelConfiguration::statusBits

Bit mask providing additional status information. [bitwise or'ed EWheelConfigStatusBits values].

1.17.2.6 float TWheelConfiguration::tireRollingCircumference

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

1.17.2.7 uint32_t TWheelConfiguration::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed EWheelConfigValidityBits values]. Must be checked before usage. Note: wheelUnit, axleIndex and wheelIndex must always be valid. Therefore no dedicated validityBits are required.

1.17.2.8 uint8_t TWheelConfiguration::wheelIndex

[Static] Identification of the location of the wheel on the axle Wheels are numbered consecutively from left to right 0: unknown/unspecified 1: left-most wheel (also used when there is only one wheel per axle, e.g. for a trike) 2: right wheel #1 (right wheel on a typical passenger car with 2 wheels per axle) 3: right wheel #2 ...

1.17.2.9 uint16_t TWheelConfiguration::wheelticksPerRevolution

[Static] Number of ticks for a single revolution of one wheel. Typically only available when wheelUnit is WHEEL_UNIT_TICKS.

1.17.2.10 EWheelUnit TWheelConfiguration::wheelUnit

[Static] Measurement unit in which the wheel rotation data is provided. WHEEL_UNIT_NONE, if no wheel rotation data is provided (and thus the rest of the structure can be ignored.

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

- wheel.h

1.18 TWheelData Struct Reference

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

Public Attributes

- uint64_t timestamp
- float data [WHEEL_MAX]
- uint32_t statusBits
- uint32_t measurementInterval
- uint32_t validityBits

1.18.1 Detailed Description

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

Wheel rotation data is provided for up to 8 wheels. The assignment of the fields data[0] ... data[7] to the wheels of the vehicle, the measurement unit and additional configuration parameters is provided as static configuration by the function snsWheelGetConfiguration(). The index used for an individual wheel in the data[] array is the same as in the TWheelConfigurationArray.

1.18.2 Member Data Documentation

1.18.2.1 float TWheelData::data[WHEEL_MAX]

Wheel rotation data for the wheel identified by TWheelConfiguration::wheel[i] in measurement unit identified by TWheelConfiguration::wheelUnit.

1.18.2.2 uint32_t TWheelData::measurementInterval

Measurement interval over which the wheel data have been acquired. Unit: micro-seconds [us]. This may slightly differ from the timestamp difference, e.g. in case of transmission jitter before timestamping. Providing the measurement interval allows thus

- a more accurate integration of wheel data measurements.
- correct usage of the first sample
- adding consistency checks

1.18.2.3 uint32_t TWheelData::statusBits

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

1.18.2.4 uint64_t TWheelData::timestamp

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

1.18.2.5 uint32_t TWheelData::validityBits

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

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

- wheel.h

2 File Documentation

2.1 acceleration.h File Reference

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

Classes

- struct TAccelerationConfiguration
- struct TAccelerationData

Typedefs

- typedef void(∗ AccelerationCallback) (const TAccelerationData accelerationData[ ], uint16_t numElements)

Enumerations

- enum EAccelerationConfigValidityBits {
ACCELERATION_CONFIG_DISTX_VALID = 0x00000001, ACCELERATION_CONFIG_DISTY_VALID = 0x00000002, ACCELERATION_CONFIG_DISTZ_VALID = 0x00000004, ACCELERATION_CONFIG_AN↩GLEYAW_VALID = 0x00000008,
ACCELERATION_CONFIG_ANGLEPITCH_VALID = 0x00000010, ACCELERATION_CONFIG_ANGLER↩OLL_VALID = 0x00000020, ACCELERATION_CONFIG_SIGMAX_VALID = 0x00000040, ACCELERATI↩ON_CONFIG_SIGMAY_VALID = 0x00000080,
ACCELERATION_CONFIG_SIGMAZ_VALID = 0x00000100, ACCELERATION_CONFIG_TYPE_VALID = 0x00000200 }
- enum EAccelerationTypeBits { ACCELERATION_X_PROVIDED = 0x00000001, ACCELERATION_Y_PR↩OVIDED = 0x00000002, ACCELERATION_Z_PROVIDED = 0x00000004, ACCELERATION_TEMPERAT↩URE_PROVIDED = 0x00000008 }
- enum EAccelerationValidityBits {
ACCELERATION_X_VALID = 0x00000001, ACCELERATION_Y_VALID = 0x00000002, ACCELERATIO↩N_Z_VALID = 0x00000004, ACCELERATION_TEMPERATURE_VALID = 0x00000008,
ACCELERATION_MEASINT_VALID = 0x00000010 }

Functions

- bool snsAccelerationInit ()
- bool snsAccelerationDestroy ()
- bool snsAccelerationGetMetaData (TSensorMetaData ∗data)
- bool snsAccelerationGetAccelerationConfiguration (TAccelerationConfiguration ∗config)
- bool snsAccelerationGetAccelerationData (TAccelerationData ∗accelerationData)
- bool snsAccelerationRegisterCallback (AccelerationCallback callback)

- bool snsAccelerationDeregisterCallback (AccelerationCallback callback)
- bool snsAccelerationGetStatus (TSensorStatus *status)
- bool snsAccelerationRegisterStatusCallback (SensorStatusCallback callback)
- bool snsAccelerationDeregisterStatusCallback (SensorStatusCallback callback)

2.1.1 Typedef Documentation

2.1.1.1 typedef void(* AccelerationCallback) (const TAccelerationData accelerationData[ ], uint16_t numElements)

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

Parameters

accelerationData	pointer to an array of TAccelerationData with size numElements
numElements	allowed range: >=1. If numElements >1, 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::angle↩Pitch.

ACCELERATION_CONFIG_ANGLEROLL_VALID Validity bit for field TAccelerationConfiguration::angle↩Roll.Validity bit for field TAccelerationConfiguration::sigmaX.

ACCELERATION_CONFIG_SIGMAX_VALID

ACCELERATION_CONFIG_SIGMAY_VALID Validity bit for field TAccelerationConfiguration::sigmaX.

ACCELERATION_CONFIG_SIGMAZ_VALID Validity bit for field TAccelerationConfiguration::sigmaZ.Validity bit for field TAccelerationConfiguration::typeBits.

ACCELERATION_CONFIG_TYPE_VALID

2.1.2.2 enum EAccelerationTypeBits

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

Enumerator

ACCELERATION_X_PROVIDED An acceleration measurement for the x-axis is provided.An acceleration measurement for the y-axis is provided.

ACCELERATION_Y_PROVIDED An acceleration measurement for the z-axis is provided.

ACCELERATION_Z_PROVIDED A measurement for the temperature is provided.

ACCELERATION_TEMPERATURE_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.

ACCELERATION_MEASINT_VALID Validity bit for field TAccelerationData::measurementInterval.

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

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.1.3.2 bool snsAccelerationDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.1.3.3 bool snsAccelerationDestroy ( )

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

Returns

True if shutdown has been successfull.

2.1.3.4 bool snsAccelerationGetAccelerationConfiguration (TAccelerationConfiguration ∗ *config*)

Accessing static configuration information about the acceleration sensor.

Parameters

config	After calling the method the currently available acceleration configuration data is written into this parameter.

Returns

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

2.1.3.5 bool snsAccelerationGetAccelerationData (TAccelerationData * *accelerationData*)

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

Parameters

accelerationData	After calling the method the currently available acceleration data is written into acceleration↩ Data.

Returns

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

2.1.3.6 bool snsAccelerationGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.1.3.7 bool snsAccelerationGetStatus (TSensorStatus * *status*)

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

Parameters

status	After calling the method the current acceleration sensor status is written into status

Returns

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

2.1.3.8 bool snsAccelerationInit ( )

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

Returns

True if initialization has been successfull.

2.1.3.9 bool snsAccelerationRegisterCallback (AccelerationCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.1.3.10 bool snsAccelerationRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.2 gyroscope.h File Reference

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

Classes

- struct TGyroscopeConfiguration
- struct TGyroscopeData

Typedefs

- typedef void(∗ GyroscopeCallback) (const TGyroscopeData gyroData[ ], uint16_t numElements)

Enumerations

- enum EGyroscopeConfigValidityBits {
  GYROSCOPE_CONFIG_ANGLEYAW_VALID = 0x00000001, GYROSCOPE_CONFIG_ANGLEPITCH_V↩ALID = 0x00000002, GYROSCOPE_CONFIG_ANGLEROLL_VALID = 0x00000004, GYROSCOPE_CON↩FIG_MOMENTYAW_VALID = 0x00000008,
  GYROSCOPE_CONFIG_SIGMAGYROSCOPE_VALID = 0x00000010, GYROSCOPE_CONFIG_TYPE_↩VALID = 0x00000020 }
- enum EGyroscopeTypeBits {
  GYROSCOPE_TEMPERATURE_COMPENSATED = 0x00000001, GYROSCOPE_YAWRATE_PROVID↩ED = 0x00000002, GYROSCOPE_PITCHRATE_PROVIDED = 0x00000004, GYROSCOPE_ROLLRATE↩_PROVIDED = 0x00000008,
  GYROSCOPE_TEMPERATURE_PROVIDED = 0x00000010 }
- enum EGyroscopeValidityBits {
  GYROSCOPE_YAWRATE_VALID = 0x00000001, GYROSCOPE_PITCHRATE_VALID = 0x00000002, G↩YROSCOPE_ROLLRATE_VALID = 0x00000004, GYROSCOPE_TEMPERATURE_VALID = 0x00000008,
  GYROSCOPE_MEASINT_VALID = 0x00000010 }

Functions

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

2.2.1 Typedef Documentation

2.2.1.1 typedef void(∗ GyroscopeCallback) (const TGyroscopeData gyroData[ ], uint16_t numElements)

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

Parameters

gyroData	pointer to an array of TGyroscopeData with size numElements
numElements	allowed range: >=1. If numElements >1, 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::momentOf↩YawInertia.

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.

GYROSCOPE_MEASINT_VALID Validity bit for field TGyroscopeData::measurementInterval.

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

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.2.3.2 bool snsGyroscopeDeregisterStatusCallback (**SensorStatusCallback** *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.2.3.3 bool snsGyroscopeDestroy ( )

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

Returns

True if shutdown has been successfull.

2.2.3.4 bool snsGyroscopeGetConfiguration (**TGyroscopeConfiguration** ∗ *gyroConfig*)

Accessing static configuration information about the gyroscope sensor.

Parameters

gyroConfig	After calling the method the currently available gyroscope configuration data is written into gyroConfig.

Returns

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

2.2.3.5 bool snsGyroscopeGetGyroscopeData (TGyroscopeData ∗ *gyroData*)

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

Parameters

gyroData	After calling the method the currently available gyroscope data is written into gyroData.

Returns

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

2.2.3.6 bool snsGyroscopeGetMetaData (TSensorMetaData ∗ *data*)

Provide meta information about sensor service.

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.2.3.7 bool snsGyroscopeGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current gyroscope status is written into status

Returns

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

2.2.3.8 bool snsGyroscopeInit ( )

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

Returns

True if initialization has been successfull.

2.2.3.9 bool snsGyroscopeRegisterCallback (GyroscopeCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.2.3.10 bool snsGyroscopeRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.3 inclination.h File Reference

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

Classes

- struct TInclinationData

Typedefs

- typedef void(∗ InclinationCallback) (const TInclinationData inclinationData[ ], uint16_t numElements)

Enumerations

- enum EInclinationSensorStatus {
  INCLINATION_INITIALIZING, INCLINATION_SENSOR_BASED, INCLINATION_UNSTABLE_DRIVING_↩
  CONDITION, INCLINATION_MODEL_BASED,
  INCLINATION_NOT_AVAILABLE }
- enum EInclinationValidityBits { INCLINATION_LONGITUDINAL_VALID = 0x00000001, INCLINATION_TR↩
  AVERSE_VALID = 0x00000002, INCLINATION_STATUS_VALID = 0x00000004 }

Functions

- bool snsInclinationInit ()
- bool snsInclinationDestroy ()
- bool snsInclinationGetMetaData (TSensorMetaData ∗data)
- bool snsInclinationGetInclinationData (TInclinationData ∗inclination)
- bool snsInclinationRegisterCallback (InclinationCallback callback)
- bool snsInclinationDeregisterCallback (InclinationCallback callback)
- bool snsInclinationGetStatus (TSensorStatus ∗status)
- bool snsInclinationRegisterStatusCallback (SensorStatusCallback callback)
- bool snsInclinationDeregisterStatusCallback (SensorStatusCallback callback)

2.3.1 Typedef Documentation

2.3.1.1 typedef void(∗ InclinationCallback) (const TInclinationData inclinationData[ ], uint16_t numElements)

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

Parameters

inclinationData	pointer to an array of TInclinationData with size numElements
numElements	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

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.3.3.2 bool snsInclinationDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.3.3.3 bool snsInclinationDestroy ( )

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

Returns

True if shutdown has been successfull.

2.3.3.4 bool snsInclinationGetInclinationData (TInclinationData ∗ *inclination*)

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

Parameters

inclination	After calling the method the currently available inclination data is written into this parameter.

Returns

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

2.3.3.5 bool snsInclinationGetMetaData (TSensorMetaData ∗ *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.3.3.6 bool snsInclinationGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current inclination sensor status is written into status

Returns

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

2.3.3.7 bool snsInclinationInit ( )

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

Returns

True if initialization has been successfull.

2.3.3.8 bool snsInclinationRegisterCallback (InclinationCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.3.3.9 bool snsInclinationRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.4 odometer.h File Reference

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

Classes

- struct TOdometerData

Typedefs

- typedef void(∗ OdometerCallback) (const TOdometerData odometerData[ ], uint16_t numElements)

Enumerations

- enum EOdometerValidityBits { ODOMETER_TRAVELLEDDISTANCE_VALID = 0x00000001 }

Functions

- bool snsOdometerInit ()
- bool snsOdometerDestroy ()
- bool snsOdometerGetMetaData (TSensorMetaData ∗data)
- bool snsOdometerGetOdometerData (TOdometerData ∗odometer)
- bool snsOdometerRegisterCallback (OdometerCallback callback)
- bool snsOdometerDeregisterCallback (OdometerCallback callback)
- bool snsOdometerGetStatus (TSensorStatus ∗status)
- bool snsOdometerRegisterStatusCallback (SensorStatusCallback callback)
- bool snsOdometerDeregisterStatusCallback (SensorStatusCallback callback)

2.4.1 Typedef Documentation

2.4.1.1 typedef void(∗ OdometerCallback) (const TOdometerData odometerData[ ], uint16_t numElements)

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

Parameters

odometerData	pointer to an array of TOdometerData with size numElements
numElements	allowed range: >=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

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.4.3.2 bool snsOdometerDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.4.3.3 bool snsOdometerDestroy ( )

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

Returns

True if shutdown has been successfull.

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

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.4.3.5 bool snsOdometerGetOdometerData (TOdometerData ∗ *odometer*)

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

Parameters

odometer	After calling the method the currently available odometer data is written into this parameter.

Returns

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

2.4.3.6 bool snsOdometerGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current odometer sensor status is written into status

Returns

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

2.4.3.7 bool snsOdometerInit ( )

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

Returns

True if initialization has been successfull.

2.4.3.8 bool snsOdometerRegisterCallback (OdometerCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.4.3.9 bool snsOdometerRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.5 reverse-gear.h File Reference

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

Classes

- struct TReverseGearData

Typedefs

- typedef void(∗ ReverseGearCallback) (const TReverseGearData reverseGearData[ ], uint16_t numElements)

Enumerations

- enum EReverseGearValidityBits { REVERSEGEAR_REVERSEGEAR_VALID = 0x00000001 }

Functions

- bool snsReverseGearInit ()
- bool snsReverseGearDestroy ()
- bool snsReverseGearGetMetaData (TSensorMetaData ∗data)
- bool snsReverseGearGetReverseGearData (TReverseGearData ∗reverseGear)
- bool snsReverseGearRegisterCallback (ReverseGearCallback callback)
- bool snsReverseGearDeregisterCallback (ReverseGearCallback callback)
- bool snsReverseGearGetStatus (TSensorStatus ∗status)
- bool snsReverseGearRegisterStatusCallback (SensorStatusCallback callback)
- bool snsReverseGearDeregisterStatusCallback (SensorStatusCallback callback)

2.5.1 Typedef Documentation

2.5.1.1 typedef void(∗ ReverseGearCallback) (const TReverseGearData reverseGearData[ ], uint16_t numElements)

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

Parameters

reverseGear← Data	pointer to an array of TReverseGearData with size numElements

numElements	allowed range: >=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

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.5.3.2 bool snsReverseGearDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.5.3.3 bool snsReverseGearDestroy ( )

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

Returns

True if shutdown has been successfull.

2.5.3.4 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

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.5.3.5 bool snsReverseGearGetReverseGearData (TReverseGearData ∗ *reverseGear*)

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

Parameters

reverseGear	After calling the method the currently available reverse gear data is written into reverseGear.

Returns

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

2.5.3.6 bool snsReverseGearGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current reverse gear sensor status is written into status

Returns

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

2.5.3.7 bool snsReverseGearInit ( )

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

Returns

True if initialization has been successfull.

2.5.3.8 bool snsReverseGearRegisterCallback (ReverseGearCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.5.3.9 bool snsReverseGearRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.6 slip-angle.h File Reference

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

Classes

- struct TSlipAngleData

Typedefs

- typedef void(∗ SlipAngleCallback) (const TSlipAngleData slipAngleData[ ], uint16_t numElements)

Enumerations

- enum ESlipAngleValidityBits { SLIPANGLE_SLIPANGLE_VALID = 0x00000001 }

Functions

- bool snsSlipAngleInit ()
- bool snsSlipAngleDestroy ()
- bool snsSlipAngleGetMetaData (TSensorMetaData ∗data)
- bool snsSlipAngleGetSlipAngleData (TSlipAngleData ∗data)
- bool snsSlipAngleRegisterCallback (SlipAngleCallback callback)
- bool snsSlipAngleDeregisterCallback (SlipAngleCallback callback)
- bool snsSlipAngleGetStatus (TSensorStatus ∗status)
- bool snsSlipAngleRegisterStatusCallback (SensorStatusCallback callback)
- bool snsSlipAngleDeregisterStatusCallback (SensorStatusCallback callback)

2.6.1 Typedef Documentation

2.6.1.1 typedef void(∗ SlipAngleCallback) (const **TSlipAngleData** slipAngleData[ ], uint16_t numElements)

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

Parameters

slipAngleData	pointer to an array of TSlipAngleData with size numElements

numElements	allowed range: >=1. If numElements >1, 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

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.6.3.2 bool snsSlipAngleDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.6.3.3 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 successfull.

2.6.3.4 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

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.6.3.5 bool snsSlipAngleGetSlipAngleData (TSlipAngleData ∗ *data*)

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

Parameters

data	After calling the method the currently available inclination data is written into this parameter.

Returns

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

2.6.3.6 bool snsSlipAngleGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current slip angle sensor status is written into status

Returns

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

2.6.3.7 bool snsSlipAngleInit ( )

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

Returns

True if initialization has been successfull.

2.6.3.8 bool snsSlipAngleRegisterCallback (SlipAngleCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.6.3.9 bool snsSlipAngleRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	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 5
- #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 5

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

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

2.7.2.3 bool snsInit ( )

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

Returns

True if initialization has been successful.

2.8 sns-meta-data.h File Reference

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

Classes

- struct TSensorMetaData

Enumerations

- enum ESensorCategory { SENSOR_CATEGORY_UNKNOWN, SENSOR_CATEGORY_LOGICAL, SENSOR_CATEGORY_PHYSICAL }
- enum ESensorType {
SENSOR_TYPE_UNKNOWN, SENSOR_TYPE_ACCELERATION, SENSOR_TYPE_GYROSCOPE, SENSOR_TYPE_INCLINATION,
SENSOR_TYPE_ODOMETER, SENSOR_TYPE_REVERSE_GEAR, SENSOR_TYPE_SLIP_ANGLE, SENSOR_TYPE_STEERING_ANGLE,
SENSOR_TYPE_VEHICLE_SPEED, SENSOR_TYPE_VEHICLE_STATE, SENSOR_TYPE_WHEELTICK, SENSOR_TYPE_WHEELSPEEDANGULAR,
SENSOR_TYPE_WHEELSPEED }

Functions

- int32_t getSensorMetaDataList (const TSensorMetaData **metadata)

2.8.1 Enumeration Type Documentation

2.8.1.1 enum ESensorCategory

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

Enumerator

SENSOR_CATEGORY_UNKNOWN Unknown category. Should not be used.

SENSOR_CATEGORY_LOGICAL A logical sensor can combine signals of several sensors.

SENSOR_CATEGORY_PHYSICAL A physical sensor provides signals from physically available sensors.

2.8.1.2 enum ESensorType

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

Enumerator

SENSOR_TYPE_UNKNOWN Unknown sensor type. Should not be used.

SENSOR_TYPE_ACCELERATION Acceleration sensor

SENSOR_TYPE_GYROSCOPE Gyroscope sensor

SENSOR_TYPE_INCLINATION Inclination sensor

SENSOR_TYPE_ODOMETER Odometer sensor

SENSOR_TYPE_REVERSE_GEAR Reverse gear sensor

SENSOR_TYPE_SLIP_ANGLE Slip angle sensor

SENSOR_TYPE_STEERING_ANGLE Steering angle sensor

SENSOR_TYPE_VEHICLE_SPEED Vehicle speed sensor

SENSOR_TYPE_VEHICLE_STATE Vehicle state sensor

SENSOR_TYPE_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

metadata	returns a a pointer an array of TSensorMetaData (maybe NULL if no metadata is available)

Returns

number of elements in the array of TSensorMetaData

2.9 sns-status.h File Reference

Classes

- struct TSensorStatus

Typedefs

- typedef void(∗ SensorStatusCallback) (const TSensorStatus ∗status)

Enumerations

- enum ESensorStatus {
SENSOR_STATUS_NOTAVAILABLE = 0, SENSOR_STATUS_INITIALIZING = 1, SENSOR_STATUS_A←
VAILABLE = 2, SENSOR_STATUS_RESTARTING = 3,
SENSOR_STATUS_FAILURE = 4, SENSOR_STATUS_OUTOFSERVICE = 5 }
- enum ESensorStatusValidityBits { SENSOR_STATUS_STATUS_VALID = 0x00000001 }

2.9.1 Typedef Documentation

2.9.1.1 typedef void(∗ SensorStatusCallback) (const TSensorStatus ∗status)

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

Parameters

status	the sensor status

2.9.2 Enumeration Type Documentation

2.9.2.1 enum ESensorStatus

Enumeration to describe the status of the sensor

Enumerator

SENSOR_STATUS_NOTAVAILABLE Sensor is not available at all, based on configuration data.

SENSOR_STATUS_INITIALIZING Initial status when the connection to the Sensor is set up for the first time.

SENSOR_STATUS_AVAILABLE Sensor is available and running as expected.

SENSOR_STATUS_RESTARTING Sensor is restarted, i.e. due to communication loss.

SENSOR_STATUS_FAILURE Sensor is not operating properly. Restarting did not help.

SENSOR_STATUS_OUTOFSERVICE Sensor is temporarily not available, due to some known external condition, vehicle bus or external ECU providing he signal being off.

2.9.2.2 enum ESensorStatusValidityBits

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

Enumerator

SENSOR_STATUS_STATUS_VALID Validity bit for field TSensorStatus::status.

2.10 steering-angle.h File Reference

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

Classes

- struct TSteeringAngleData
- struct TSteeringAngleConfiguration

Typedefs

- typedef void(∗ SteeringAngleCallback) (const TSteeringAngleData steeringAngleData[ ], uint16_t num↩ Elements)

Enumerations

- enum ESteeringAngleValidityBits { STEERINGANGLE_FRONT_VALID = 0x00000001, STEERINGANGL↩ E_REAR_VALID = 0x00000002, STEERINGANGLE_STEERINGWHEEL_VALID = 0x00000004 }

Functions

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

2.10.1 Typedef Documentation

2.10.1.1 typedef void(∗ SteeringAngleCallback) (const TSteeringAngleData steeringAngleData[ ], uint16_t numElements)

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

Parameters

steeringAngleData	pointer to an array of TSteeringAngleData with size numElements
numElements	allowed range: >=1. If numElements >1, buffered data are provided.

2.10.2 Enumeration Type Documentation

2.10.2.1 enum ESteeringAngleValidityBits

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

Enumerator

STEERINGANGLE_FRONT_VALID Validity bit for field TSteeringAngleData::front.

STEERINGANGLE_REAR_VALID Validity bit for field TSteeringAngleData::rear.

STEERINGANGLE_STEERINGWHEEL_VALID Validity bit for field TSteeringAngleData::steeringWheel.

2.10.3 Function Documentation

2.10.3.1 bool snsSteeringAngleDeregisterCallback (SteeringAngleCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.10.3.2 bool snsSteeringAngleDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.10.3.3 bool snsSteeringAngleDestroy ( )

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

Returns

True if shutdown has been successfull.

2.10.3.4 bool snsSteeringAngleGetConfiguration (TSteeringAngleConfiguration ∗ *config*)

Accessing static configuration information about the steering angle sensor.

Parameters

config	After calling the method the currently available static steering angle configuration data is written into config.

Returns

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

2.10.3.5 bool snsSteeringAngleGetMetaData (TSensorMetaData ∗ *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.10.3.6 bool snsSteeringAngleGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current steering angle sensor status is written into status

Returns

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

2.10.3.7 bool snsSteeringAngleGetSteeringAngleData (TSteeringAngleData ∗ *angleData*)

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

Parameters

angleData	After calling the method the currently available steering angle data is written into angleData.

Returns

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

2.10.3.8 bool snsSteeringAngleInit ( )

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

Returns

True if initialization has been successfull.

2.10.3.9 bool snsSteeringAngleRegisterCallback (SteeringAngleCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.10.3.10 bool snsSteeringAngleRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.11 vehicle-data.h File Reference

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

Classes

- struct TDistance3D
- struct TVehicleDataConfiguration

Enumerations

- enum EAxleType { SNS_AXLE_UNKNOWN = 0, SNS_AXLE_FRONT = 1, SNS_AXLE_REAR = 2, SNS_↵ AXLE_BOTH = 3 }
- enum EVehicleType { SNS_CAR = 0, SNS_TRUCK = 1, SNS_MOTORBIKE = 2, SNS_BUS = 3 }
- enum EVehicleDataConfigurationValidityBits {
VEHICLESTATUS_VEHICLETYPE_VALID = 0x00000001, VEHICLESTATUS_DRIVENAXLES_VALID = 0x00000002, VEHICLESTATUS_SEATCOUNT_VALID = 0x00000004, VEHICLESTATUS_TRACKWIDT↵ H_VALID = 0x00000008,
VEHICLESTATUS_FRONTAXLETRACKWIDTH_VALID = 0x00000010, VEHICLESTATUS_WHELLBAS↵ E_VALID = 0x00000020, VEHICLESTATUS_VEHICLEMASS_VALID = 0x00000040, VEHICLESTATUS_↵ VEHICLEWIDTH_VALID = 0x00000080,
VEHICLESTATUS_DISTCOG2REFPOINT_VALID = 0x00000100, VEHICLESTATUS_DISTFRONT↵ AXLE2REFPOINT_VALID = 0x00000200, VEHICLESTATUS_DISTREARAXLE2REFPOINT_VALID = 0x00000400 }

Functions

- bool snsVehicleDataInit ()
- bool snsVehicleDataDestroy ()
- bool snsVehicleDataGetConfiguration (TVehicleDataConfiguration ∗vehicleDataConfiguration)

2.11.1 Enumeration Type Documentation

2.11.1.1 enum EAxleType

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

Enumerator

SNS_AXLE_UNKNOWN It is not known which axles are driven.

SNS_AXLE_FRONT Only the front axle is driven.

SNS_AXLE_REAR Only the rear axle is driven.

SNS_AXLE_BOTH Both axles are driven (4 wheel drive).

2.11.1.2 enum EVehicleDataConfigurationValidityBits

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

Enumerator

VEHICLESTATUS_VEHICLETYPE_VALID Validity bit for field TVehicleDataConfiguration::vehicleType.

VEHICLESTATUS_DRIVENAXLES_VALID Validity bit for field TVehicleDataConfiguration::drivenAxles.

VEHICLESTATUS_SEATCOUNT_VALID Validity bit for field TVehicleDataConfiguration::seatCount.

VEHICLESTATUS_TRACKWIDTH_VALID Validity bit for field TVehicleDataConfiguration::trackWidth.

VEHICLESTATUS_FRONTAXLETRACKWIDTH_VALID Validity bit for field TVehicleDataConfiguration↵ ::frontAxleTrackWidth.

VEHICLESTATUS_WHELLBASE_VALID Validity bit for field TVehicleDataConfiguration::wheelBase.

VEHICLESTATUS_VEHICLEMASS_VALID Validity bit for field TVehicleDataConfiguration::vehicleMass.

VEHICLESTATUS_VEHICLEWIDTH_VALID Validity bit for field TVehicleDataConfiguration::vehicleWidth.

VEHICLESTATUS_DISTCOG2REFPOINT_VALID Validity bit for field TVehicleDataConfiguration::distCo↵ G2RefPoint.

VEHICLESTATUS_DISTFRONTAXLE2REFPOINT_VALID Validity bit for field TVehicleDataConfiguration::distFrontAxle2RefPoint.

VEHICLESTATUS_DISTREARAXLE2REFPOINT_VALID Validity bit for field TVehicleDataConfiguration::vehicleType.

2.11.1.3 enum EVehicleType

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

Enumerator

SNS_CAR Passenger car with 4 wheels.

SNS_TRUCK Truck

SNS_MOTORBIKE Motorbike with 2 wheels.

SNS_BUS Passenger bus.

2.11.2 Function Documentation

2.11.2.1 bool snsVehicleDataDestroy ( )

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

Returns

True if shutdown has been successfull.

2.11.2.2 bool snsVehicleDataGetConfiguration (TVehicleDataConfiguration ∗ *vehicleDataConfiguration*)

Accessing static configuration information about the vehicle.

Parameters

vehicleDataConfiguration	After calling the method the available vehicle configuration data is written into vehicleData.

Returns

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

2.11.2.3 bool snsVehicleDataInit ( )

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

Returns

True if initialization has been successfull.

2.12 vehicle-speed.h File Reference

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

Classes

- struct TVehicleSpeedData

Typedefs

- typedef void(∗ VehicleSpeedCallback) (const TVehicleSpeedData vehicleSpeedData[ ], uint16_t num↵ Elements)

Enumerations

- enum EVehicleSpeedValidityBits { VEHICLESPEED_VEHICLESPEED_VALID = 0x00000001, VEHICLES↵ PEED__MEASINT_VALID = 0x00000002 }

Functions

- bool snsVehicleSpeedInit ()
- bool snsVehicleSpeedDestroy ()
- bool snsVehicleSpeedGetMetaData (TSensorMetaData ∗data)
- bool snsVehicleSpeedGetVehicleSpeedData (TVehicleSpeedData ∗vehicleSpeed)
- bool snsVehicleSpeedRegisterCallback (VehicleSpeedCallback callback)
- bool snsVehicleSpeedDeregisterCallback (VehicleSpeedCallback callback)
- bool snsVehicleSpeedGetStatus (TSensorStatus ∗status)
- bool snsVehicleSpeedRegisterStatusCallback (SensorStatusCallback callback)
- bool snsVehicleSpeedDeregisterStatusCallback (SensorStatusCallback callback)

2.12.1 Typedef Documentation

2.12.1.1 typedef void(∗ VehicleSpeedCallback) (const **TVehicleSpeedData** vehicleSpeedData[ ], uint16_t numElements)

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

Parameters

vehicleSpeed↵ Data	pointer to an array of TVehicleSpeedData with size numElements
numElements	allowed range: >=1. If numElements >1, buffered data are provided.

2.12.2 Enumeration Type Documentation

2.12.2.1 enum **EVehicleSpeedValidityBits**

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

Enumerator

VEHICLESPEED_VEHICLESPEED_VALID Validity bit for field TVehicleSpeedData::vehicleSpeed.

VEHICLESPEED__MEASINT_VALID Validity bit for field TVehicleSpeedData::measurementInterval.

2.12.3 Function Documentation

2.12.3.1 bool snsVehicleSpeedDeregisterCallback (VehicleSpeedCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.12.3.2 bool snsVehicleSpeedDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.12.3.3 bool snsVehicleSpeedDestroy ( )

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

Returns

True if shutdown has been successfull.

2.12.3.4 bool snsVehicleSpeedGetMetaData (TSensorMetaData ∗ *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.12.3.5 bool snsVehicleSpeedGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current vehicle speed sensor status is written into status

Returns

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

2.12.3.6 bool snsVehicleSpeedGetVehicleSpeedData (TVehicleSpeedData ∗ *vehicleSpeed*)

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

Parameters

vehicleSpeed	After calling the method the currently available vehicle speed data is written into vehicle↩ Speed.

Returns

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

2.12.3.7 bool snsVehicleSpeedInit ( )

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

Returns

True if initialization has been successfull.

2.12.3.8 bool snsVehicleSpeedRegisterCallback (VehicleSpeedCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.12.3.9 bool snsVehicleSpeedRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.13 vehicle-state.h File Reference

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

Classes

- struct TVehicleStateData

Typedefs

- typedef void(∗ VehicleStateCallback) (const TVehicleStateData vehicleState[ ], uint16_t numElements)

Enumerations

- enum EVehicleStateValidityBits { VEHICLESTATE_ANTILOCKBRAKESYSTEMACTIVE_VALID = 0x00000001, VEHICLESTATE_BRAKEACTIVE_VALID = 0x00000002, VEHICLESTATE_ELECTRONICSTABILIT↩ YPROGRAMACTIVE_VALID = 0x00000004, VEHICLESTATE_TRACTIONCONTROLACTIVE_VALID = 0x00000008 }

Functions

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

2.13.1 Typedef Documentation

2.13.1.1 typedef void(* VehicleStateCallback) (const **TVehicleStateData** vehicleState[ ], uint16_t numElements)

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

Parameters

vehicleSpeed↩ Data	pointer to an array of TVehicleStateData with size numElements
numElements	allowed range: >=1. If numElements >1, buffered data are provided.

2.13.2 Enumeration Type Documentation

2.13.2.1 enum **EVehicleStateValidityBits**

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

Enumerator

VEHICLESTATE_ANTILOCKBRAKESYSTEMACTIVE_VALID Validity bit for field TVehicleStateData::anti↩ LockBrakeSystemActive.

VEHICLESTATE_BRAKEACTIVE_VALID Validity bit for field TVehicleStateData::brakeActive.

VEHICLESTATE_ELECTRONICSTABILITYPROGRAMACTIVE_VALID Validity bit for field TVehicleState↩ Data::electronicStabilityProgramActive.

VEHICLESTATE_TRACTIONCONTROLACTIVE_VALID Validity bit for field TVehicleStateData::traction↩ ControlActive.

2.13.3 Function Documentation

2.13.3.1 bool snsVehicleStateDeregisterCallback (**VehicleStateCallback** *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.13.3.2 bool snsVehicleStateDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.13.3.3 bool snsVehicleStateDestroy ( )

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

Returns

True if shutdown has been successfull.

2.13.3.4 bool snsVehicleStateGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.13.3.5 bool snsVehicleStateGetStatus (TSensorStatus * *status*)

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

Parameters

status	After calling the method the current vehicle state sensor status is written into status

Returns

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

2.13.3.6 bool snsVehicleStateGetVehicleStateData (TVehicleStateData * *vehicleState*)

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

Parameters

vehicleState	After calling the method the currently available vehicle state data is written into vehicleState.

Returns

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

2.13.3.7 bool snsVehicleStateInit ( )

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

Returns

True if initialization has been successfull.

2.13.3.8 bool snsVehicleStateRegisterCallback (VehicleStateCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.13.3.9 bool snsVehicleStateRegisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.14 wheel.h File Reference

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

Classes

- struct TWheelConfiguration
- struct TWheelData

Macros

- #define WHEEL_MAX 8

Typedefs

- typedef TWheelConfiguration TWheelConfigurationArray[WHEEL_MAX]
- typedef void(∗ WheelCallback) (const TWheelData wheelData[ ], uint16_t numElements)
- typedef void(∗ WheelConfigurationCallback) (const TWheelConfigurationArray ∗config)

Enumerations

- enum EWheelStatusBits { WHEEL_STATUS_GAP = 0x00000001, WHEEL_STATUS_INIT = 0x00000002 }
- enum EWheelValidityBits {
  WHEEL0_VALID = 0x00000001, WHEEL1_VALID = 0x00000002, WHEEL2_VALID = 0x00000004, WHEEL3_VALID = 0x00000008,
  WHEEL4_VALID = 0x00000010, WHEEL5_VALID = 0x00000020, WHEEL6_VALID = 0x00000040, WHEEL7_VALID = 0x00000080,
  WHEEL_MEASINT_VALID = 0x00000100 }
- enum EWheelUnit { WHEEL_UNIT_NONE = 0, WHEEL_UNIT_TICKS = 1, WHEEL_UNIT_SPEED = 2, WHEEL_UNIT_ANGULAR_SPEED = 3 }
- enum EWheelConfigStatusBits { WHEEL_CONFIG_DRIVEN = 0x00000001, WHEEL_CONFIG_STEERED = 0x00000002, WHEEL_CONFIG_DIFF_LOCK = 0x00000004 }
- enum EWheelConfigValidityBits {
  WHEEL_CONFIG_TICKS_PER_REV_VALID = 0x00000001, WHEEL_CONFIG_TIRE_CIRC_VALID = 0x00000002, WHEEL_CONFIG_DISTX_VALID = 0x00000004, WHEEL_CONFIG_DISTY_VALID = 0x00000008,
  WHEEL_CONFIG_DISTZ_VALID = 0x00000010, WHEEL_CONFIG_DRIVEN_VALID = 0x00000020, WHEEL_CONFIG_STEERED_VALID = 0x00000040, WHEEL_CONFIG_DIFF_LOCK_VALID = 0x00000080
  }

Functions

- bool snsWheelInit ()
- bool snsWheelDestroy ()
- bool snsWheelGetMetaData (TSensorMetaData ∗data)
- bool snsWheelGetConfiguration (TWheelConfigurationArray ∗config)
- bool snsWheelRegisterConfigurationCallback (WheelConfigurationCallback callback)
- bool snsWheelDeregisterConfigurationCallback (WheelConfigurationCallback callback)
- bool snsWheelGetWheelData (TWheelData ∗wheelData)
- bool snsWheelRegisterCallback (WheelCallback callback)
- bool snsWheelDeregisterCallback (WheelCallback callback)
- bool snsWheelGetStatus (TSensorStatus ∗status)
- bool snsWheelRegisterStatusCallback (SensorStatusCallback callback)
- bool snsWheelDeregisterStatusCallback (SensorStatusCallback callback)

2.14.1 Macro Definition Documentation

2.14.1.1 #define WHEEL_MAX 8

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

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

Note: The driving direction (forward/backward) shall always be coded as sign of the wheel rotation data. This will reduce synchronization effort with separate reverse gear info by the client application which is in particular useful with buffered data. How the driving direction can be derived is vehicle specific. Maximum number of wheel elements per structure. A fix value is used because a flat data structure has advantages like simple copying, indexed access.

2.14.2 Typedef Documentation

2.14.2.1 typedef **TWheelConfiguration** TWheelConfigurationArray[**WHEEL_MAX**]

Set of configuration data for all wheels of the vehicle. The index of configuration data for an individual wheel in the array is fixed during the runtime of the system. Unused fields, i.e. those for which wheelUnit is WHEEL_UNIT_N↩ONE will be at the tail of the array.

2.14.2.2 typedef void(∗ WheelCallback) (const **TWheelData** wheelData[ ], uint16_t numElements)

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

Parameters

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

2.14.2.3 typedef void(∗ WheelConfigurationCallback) (const **TWheelConfigurationArray** ∗config)

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

Parameters

config	the updated wheel configuration

2.14.3 Enumeration Type Documentation

2.14.3.1 enum **EWheelConfigStatusBits**

Wheel configuration status bits

Enumerator

- ***WHEEL_CONFIG_DRIVEN*** The wheel is driven by the powertrain. It may thus be affected by additional wheel slip.
- ***WHEEL_CONFIG_STEERED*** The wheel may be turned by the steering. This is typically the case only for wheels on the front axle. But for some vehicles also wheels on other axles are permanently or temporarily steered.
- ***WHEEL_CONFIG_DIFF_LOCK*** The differential lock for this wheel is activated.

2.14.3.2 enum **EWheelConfigValidityBits**

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

Enumerator

- ***WHEEL_CONFIG_TICKS_PER_REV_VALID*** Validity bit for field TWheelConfiguration::wheelticksPer↩Revolution.
- ***WHEEL_CONFIG_TIRE_CIRC_VALID*** Validity bit for field TWheelConfiguration::tireRollingCircumference.
- ***WHEEL_CONFIG_DISTX_VALID*** Validity bit for field TWheelConfiguration::dist2RefPointX.
- ***WHEEL_CONFIG_DISTY_VALID*** Validity bit for field TWheelConfiguration::dist2RefPointY.

WHEEL_CONFIG_DISTZ_VALID Validity bit for field TWheelConfiguration::dist2RefPointZ.

WHEEL_CONFIG_DRIVEN_VALID Validity bit for field TWheelConfiguration::EWheelConfigStatusBits::W↩ HEEL_CONFIG_DRIVEN.

WHEEL_CONFIG_STEERED_VALID Validity bit for field TWheelConfiguration::EWheelConfigStatusBits::↩ WHEEL_CONFIG_STEERED.

WHEEL_CONFIG_DIFF_LOCK_VALID Validity bit for field TWheelConfiguration::EWheelConfigStatusBits↩ ::WHEEL_CONFIG_DIFF_LOCK.

2.14.3.3 enum EWheelStatusBits

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

Background information

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

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

Further Background

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

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

Enumerator

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

- wheel tick data on the vehicle bus explicitly tagged as invalid
- interrupted reception of vehicle bus messages. This flag will only be set if the detected gap may affect the application.

WHEEL_STATUS_INIT This is the first wheel data of a bus or ignition lifecycle, i.e. the wheel tick difference calculation may be less reliable.

2.14.3.4 enum EWheelUnit

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

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

Enumerator

WHEEL_UNIT_NONE Wheel does not provide any data.

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

WHEEL_UNIT_SPEED Wheel rotation data is provided as speed in [m/s].

WHEEL_UNIT_ANGULAR_SPEED Wheel rotation data is provided as angular speed in [1/s] rotation per seconds.

2.14.3.5 enum EWheelValidityBits

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

Enumerator

WHEEL0_VALID Validity bit for field TWheelData::data[0].

WHEEL1_VALID Validity bit for field TWheelData::data[1].

WHEEL2_VALID Validity bit for field TWheelData::data[2].

WHEEL3_VALID Validity bit for field TWheelData::data[3].

WHEEL4_VALID Validity bit for field TWheelData::data[4].

WHEEL5_VALID Validity bit for field TWheelData::data[5].

WHEEL6_VALID Validity bit for field TWheelData::data[6].

WHEEL7_VALID Validity bit for field TWheelData::data[7].

WHEEL_MEASINT_VALID Validity bit for field TWheelData::measurementInterval.

2.14.4 Function Documentation

2.14.4.1 bool snsWheelDeregisterCallback (**WheelCallback** *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully. Note: This function will only return false is the callback was not registered.

2.14.4.2 bool snsWheelDeregisterConfigurationCallback (**WheelConfigurationCallback** *callback*)

Deregister wheel configuration callback. After returning from this method no new wheel configuration updates will be delivered to the client.

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.14.4.3 bool snsWheelDeregisterStatusCallback (SensorStatusCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.14.4.4 bool snsWheelDestroy ( )

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

Returns

True if shutdown has been successful. Note: In case the shutdown has not been successful, a retry does not make sense. The return value is intended primarily for diagnostics.

2.14.4.5 bool snsWheelGetConfiguration (TWheelConfigurationArray ∗ *config*)

Accessing static configuration information about the wheel sensor.

Parameters

config	After calling the method the current wheel configuration is written into this parameter. Note: as some parts of the wheel configuration may change during runtime it is recommended to register for configuration updates.

Returns

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

2.14.4.6 bool snsWheelGetMetaData (TSensorMetaData ∗ *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.14.4.7 bool snsWheelGetStatus (TSensorStatus ∗ *status*)

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

Parameters

status	After calling the method the current wheel sensor status is written into status

Returns

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

2.14.4.8 bool snsWheelGetWheelData (TWheelData ∗ *wheelData*)

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

Parameters

wheelData	After calling the method the currently available wheel rotation data is written into the array.

Returns

Is true if data can be provided and false otherwise, e.g. missing initialization Note: Wheel rotation data typically changes while the vehicle is moving. Therefore for most applications it is better to register for wheel rotation data updates.

2.14.4.9 bool snsWheelInit ( )

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

Returns

True if initialization has been successful. Note: In case the initialization has not been successful during system startup, a later retry may be successful.

2.14.4.10 bool snsWheelRegisterCallback (WheelCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully. Note: The function will only return false if either snsWheel←Init() has not been called before or if the number of concurrently supported callbacks is exceeded.

2.14.4.11 bool snsWheelRegisterConfigurationCallback (WheelConfigurationCallback *callback*)

Register wheel configuration callback. This is the recommended method for continuously monitoring the wheel configuration. The callback will be invoked when updated wheel configuration is available.

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully. Note: The function will only return false if either snsWheel←Init() has not been called before or if the number of concurrently supported callbacks is exceeded.

2.14.4.12 bool snsWheelRegisterStatusCallback (SensorStatusCallback *callback*)

Register wheel sensor status callback. This is the recommended method for continuously monitoring the wheel sensor status. The callback will be invoked when the wheel sensor status has changed.

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully. Note: The function will only return false if either snsWheel↵ Init() has not been called before or if the number of concurrently supported callbacks is exceeded.

Index