GENIVI SensorsService

Component Specification

Release 3.0.0-Alpha
Status: Draft
24.04.2014

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

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

Keywords:
SensorsService, Sensors, Positioning API.

Table of Contents

Change History

Version	Date	Author	Change
0.1	27.08.2013	MResidori	Document Created
0.2	18.11.2013	MResidori	Document generated from the Entrprise Architect Model
0.3	27.03.2014	MResidori	Added copyright notes
3.0.0-alpha	24.04.2014	MResidori	Changed license version from 3.0 to 4.0

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 GNSSService is typically implemented as a C library that is dynamically linked by the EnhancedPositionService.

2. SensorsService Diagram

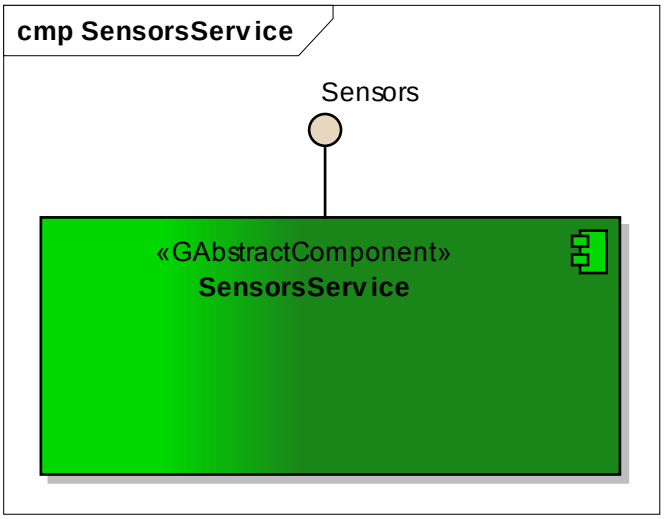

Figure: 2

3. Traceability Diagram

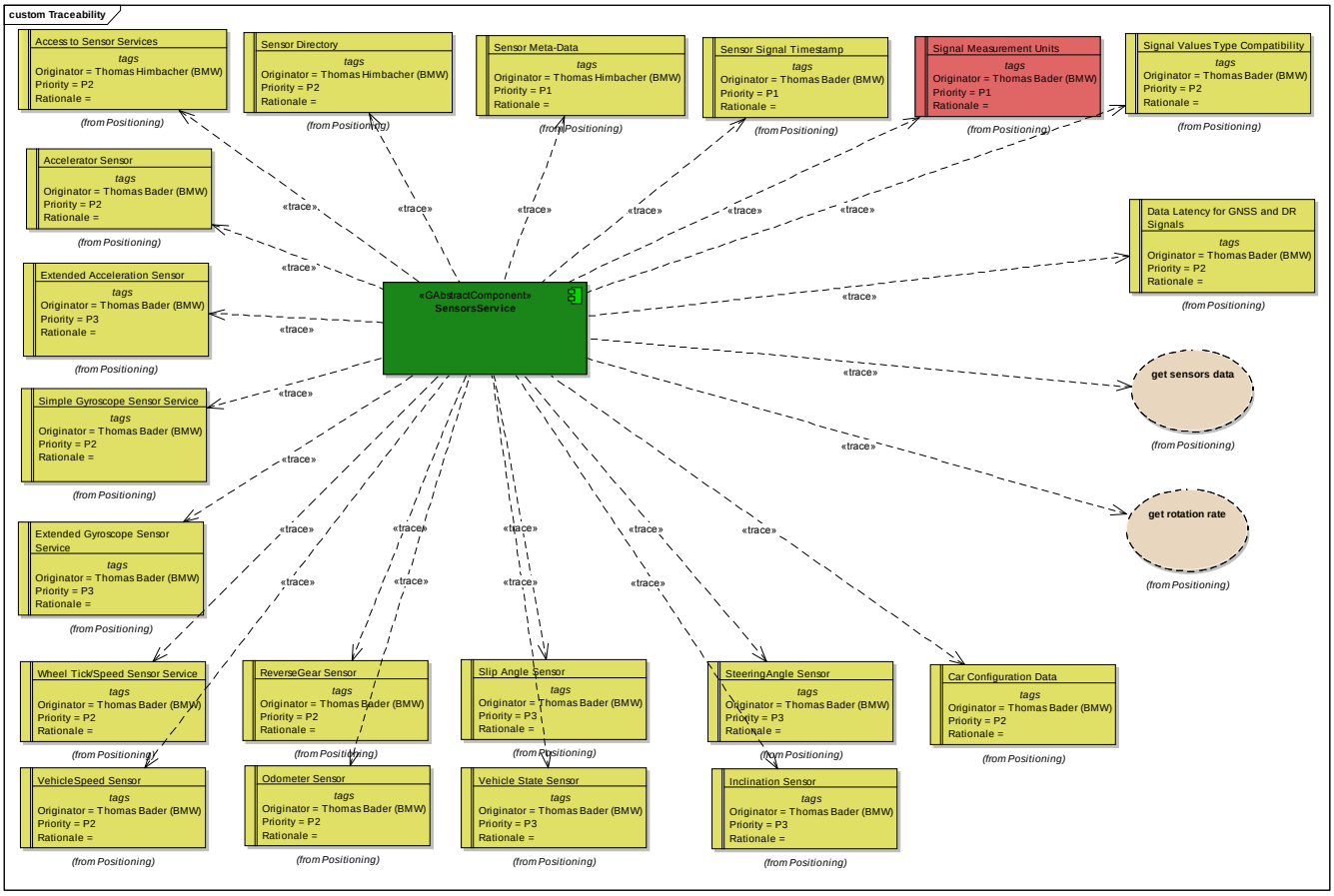

Figure: 3

1. *Context*

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

2. Context Diagram

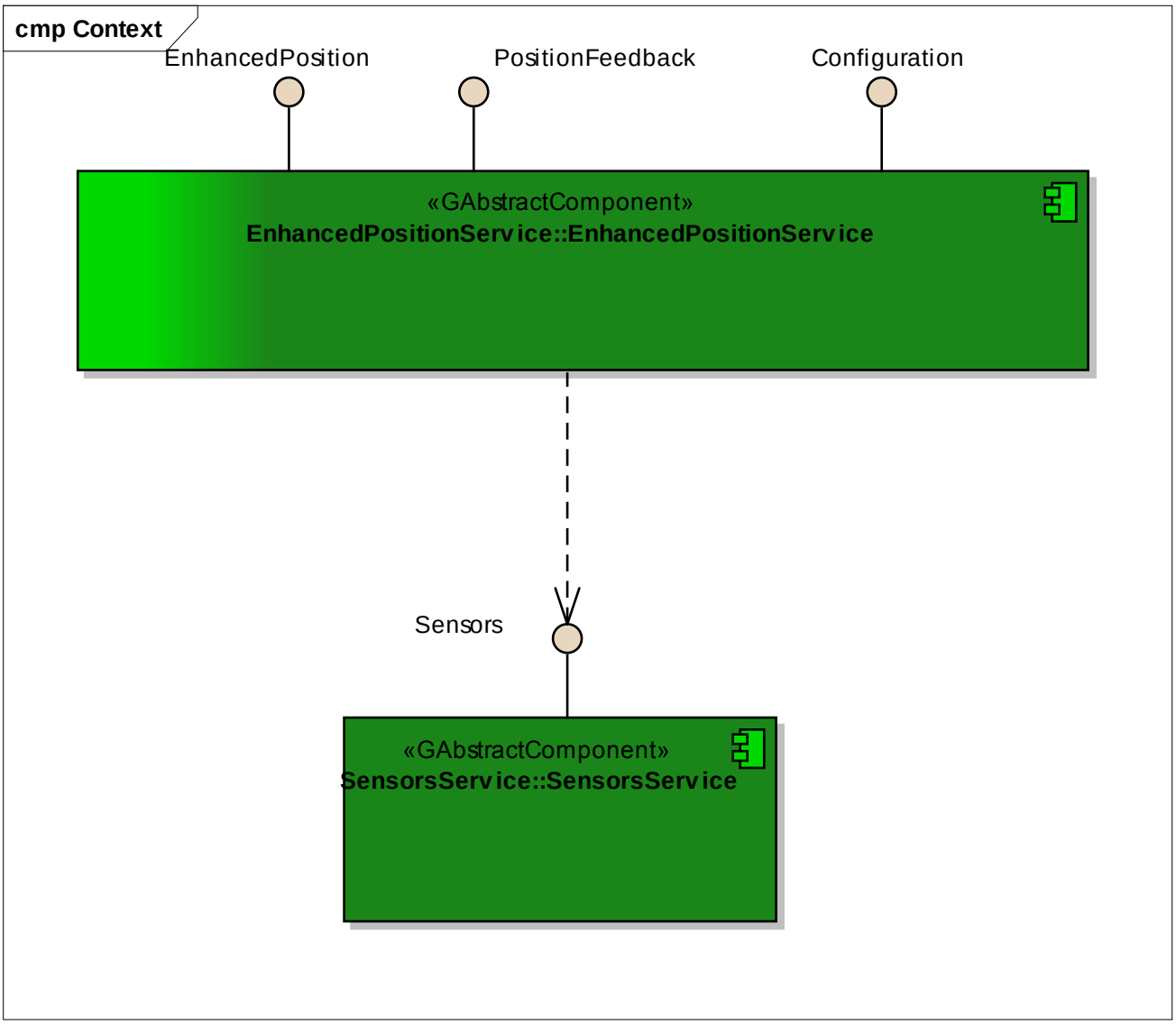

Figure: 4

SensorsService

Generated by Doxygen 1.7.6.1

Thu Apr 24 2014 17:27:47

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
- EAccelerationTypeBits typeBits
- uint32_t validityBits

1.1.1 Detailed Description

Static configuration data for the acceleration sensor service.

BEGIN Explanation of the angleYaw, anglePitch angleRoll parameters The orientation of the accelerometer hardware (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 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 EAccelerationTypeBits TAccelerationConfiguration::typeBits

Bit mask indicating the type of the used gyroscope.

1.1.2.11 uint32_t TAccelerationConfiguration::validityBits

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

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

- acceleration.h

1.2 TAccelerationData Struct Reference

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

Public Attributes

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

1.2.1 Detailed Description

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

See also

TAccelerationConfiguration for an explanation how to convert these values to the vehicle axis system

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

1.2.2 Member Data Documentation

1.2.2.1 float TAccelerationData::temperature

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

1.2.2.2 uint64_t TAccelerationData::timestamp

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

1.2.2.3 uint32_t TAccelerationData::validityBits

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

1.2.2.4 float TAccelerationData::x

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

1.2.2.5 float TAccelerationData::y

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

1.2.2.6 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.

1.3.2 Member Data Documentation

1.3.2.1 float TDistance3D::x

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

1.3.2.2 float TDistance3D::y

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

1.3.2.3 float TDistance3D::z

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

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

- vehicle-data.h

1.4 TGyroscopeConfiguration Struct Reference

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

Public Attributes

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

1.4.1 Detailed Description

Static configuration data for the gyroscope sensor service.

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

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

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

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

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

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

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

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

1.4.2 Member Data Documentation

1.4.2.1 float TGyroscopeConfiguration::anglePitch

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

1.4.2.2 float TGyroscopeConfiguration::angleRoll

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

1.4.2.3 float TGyroscopeConfiguration::angleYaw

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

1.4.2.4 float TGyroscopeConfiguration::momentOfYawInertia

Moment of yaw inertia. The pitch and roll inertia moments are not provided as they are not relevant for positioning. [kg∗m^2]

1.4.2.5 float TGyroscopeConfiguration::sigmaGyroscope

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

1.4.2.6 EGyroscopeTypeBits TGyroscopeConfiguration::typeBits

Bit mask indicating the type of the used gyroscope.

1.4.2.7 uint32_t TGyroscopeConfiguration::validityBits

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

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

- gyroscope.h

1.5 TGyroscopeData Struct Reference

`#include <gyroscope.h>`

Public Attributes

- uint64_t timestamp
- float yawRate
- float pitchRate
- float rollRate
- float temperature
- uint32_t validityBits

1.5.1 Detailed Description

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

See also

TGyroscopeConfiguration for an explanation how to convert these values to the vehicle axis system

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

1.5.2 Member Data Documentation

1.5.2.1 float TGyroscopeData::pitchRate

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

1.5.2.2 float TGyroscopeData::rollRate

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

1.5.2.3 float TGyroscopeData::temperature

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

1.5.2.4 uint64_t TGyroscopeData::timestamp

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

1.5.2.5 uint32_t TGyroscopeData::validityBits

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

1.5.2.6 float TGyroscopeData::yawRate

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

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

- gyroscope.h

1.6 TInclinationData Struct Reference

`#include <inclination.h>`

Public Attributes

- uint64_t timestamp
- float longitudinalGradientRoadway
- float traverseGradientRoadway
- EInclinationSensorStatus status
- uint32_t validityBits

1.6.1 Detailed Description

Inclination sensor service provides the inclination values.

1.6.2 Member Data Documentation

1.6.2.1 float TInclinationData::longitudinalGradientRoadway

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

1.6.2.2 EInclinationSensorStatus TInclinationData::status

Status of the signals.

1.6.2.3 uint64_t TInclinationData::timestamp

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

1.6.2.4 float TInclinationData::traverseGradientRoadway

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

1.6.2.5 uint32_t TInclinationData::validityBits

Bit mask indicating the validity of each corresponding value. [bitwise or'ed EInclination-ValidityBits 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 EOdometer-ValidityBits 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 EReverse-GearValidityBits values]. Must be checked before usage.

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

- reverse-gear.h

1.9 TSensorMetaData Struct Reference

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

Public Attributes

- uint32_t version
- ESensorCategory category
- ESensorType type
- uint32_t cycleTime

1.9.1 Detailed Description

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

1.9.2 Member Data Documentation

1.9.2.1 ESensorCategory TSensorMetaData::category

Sensor Category (Physical/Logical).

1.9.2.2 uint32_t TSensorMetaData::cycleTime

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

1.9.2.3 ESensorType TSensorMetaData::type

Sensor Type (Odometer, Gyroscope, etc.).

1.9.2.4 uint32_t TSensorMetaData::version

Version of the sensor service.

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

- sns-meta-data.h

1.10 TSlipAngleData Struct Reference

`#include <slip-angle.h>`

Public Attributes

- uint64_t timestamp

- float slipAngle
- uint32_t validityBits

1.10.1 Detailed Description

Slip angle sensor service provides the slip angle value. The reference coordinate system for the slip angle is defined here: https://collab.genivi.org/wiki/display/genivi/LBSSensorServiceRequirements-Borg#LBSSensorServiceRequirementsBorg-ReferenceSystem

1.10.2 Member Data Documentation

1.10.2.1 float TSlipAngleData::slipAngle

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

1.10.2.2 uint64_t TSlipAngleData::timestamp

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

1.10.2.3 uint32_t TSlipAngleData::validityBits

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

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

- slip-angle.h

1.11 TSteeringAngleConfiguration Struct Reference

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

Public Attributes

- float sigmaSteeringAngle
- float sigmaSteeringWheelAngle
- float steeringRatio

1.11.1 Detailed Description

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

1.11.2 Member Data Documentation

1.11.2.1 float TSteeringAngleConfiguration::sigmaSteeringAngle

Standard deviation of the steering front angle in [degree].

1.11.2.2 float TSteeringAngleConfiguration::sigmaSteeringWheelAngle

Standard deviation of the steering wheel angle in [degree].

1.11.2.3 float TSteeringAngleConfiguration::steeringRatio

Steering ratio between steering wheel and wheels. Only valid when static: Unit: [-]

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

- steering-angle.h

1.12 TSteeringAngleData Struct Reference

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

Public Attributes

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

1.12.1 Detailed Description

The SteeringAngle delivers the sensor values of the steering angle. You must check the valid bits before usage.

1.12.2 Member Data Documentation

1.12.2.1 float TSteeringAngleData::front

Returns the steering angle of the front wheels [degree].

1.12.2.2 float TSteeringAngleData::rear

Returns the steering angle of the rear wheels [degree]. This is only useful for vehicles with a steerable rear axis.

1.12.2.3 float TSteeringAngleData::steeringWheel

Returns the angle of the steering wheel [degree]. Useful for vehicles where the wheel angles are not available. Must be used in combination with the steeringRatio.

1.12.2.4 uint64_t TSteeringAngleData::timestamp

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

1.12.2.5 uint32_t TSteeringAngleData::validityBits

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

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

- steering-angle.h

1.13 TVehicleSpeedData Struct Reference

`#include <vehicle-speed.h>`

Public Attributes

- uint64_t timestamp
- float vehicleSpeed
- uint32_t validityBits

1.13.1 Detailed Description

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

1.13.2 Member Data Documentation

1.13.2.1 uint64_t TVehicleSpeedData::timestamp

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

1.13.2.2 uint32_t TVehicleSpeedData::validityBits

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

1.13.2.3 float TVehicleSpeedData::vehicleSpeed

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

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

- vehicle-speed.h

1.14 TVehicleStateData Struct Reference

`#include <vehicle-state.h>`

Public Attributes

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

1.14.1 Detailed Description

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

1.14.2 Member Data Documentation

1.14.2.1 bool TVehicleStateData::antiLockBrakeSystemActive

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

1.14.2.2 bool TVehicleStateData::brakeActive

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

1.14.2.3 bool TVehicleStateData::electronicStabilityProgramActive

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

1.14.2.4 uint64_t TVehicleStateData::timestamp

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

1.14.2.5 bool TVehicleStateData::tractionControlActive

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

1.14.2.6 uint32_t TVehicleStateData::validityBits

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

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

- vehicle-state.h

1.15 TWheelspeed Struct Reference

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

Public Attributes

- uint64_t timestamp
- TWheelspeedDetail elements [WHEEL_NUM_ELEMENTS]

1.15.1 Detailed Description

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

1.15.2 Member Data Documentation

1.15.2.1 TWheelspeedDetail TWheelspeed::elements[WHEEL_NUM_ELEMENTS]

The array of wheel speed elements.

1.15.2.2 uint64_t TWheelspeed::timestamp

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

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

- wheel.h

1.16 TWheelspeedAngular Struct Reference

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

Public Attributes

- uint64_t timestamp
- TWheelspeedAngularDetail elements [WHEEL_NUM_ELEMENTS]

1.16.1 Detailed Description

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

1.16.2 Member Data Documentation

1.16.2.1 TWheelspeedAngularDetail TWheelspeedAngular::elements[WHEEL_NUM_ELEMENTS]

The array of angular wheel speed elements.

1.16.2.2 uint64_t TWheelspeedAngular::timestamp

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

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

- wheel.h

1.17 TWheelspeedAngularDetail Struct Reference

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

Public Attributes

- EWheelId id
- float rotations

1.17.1 Detailed Description

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

1.17.2 Member Data Documentation

1.17.2.1 EWheelId TWheelspeedAngularDetail::id

1.17.2.2 float TWheelspeedAngularDetail::rotations

unit is [1/s] rotation per seconds

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

- wheel.h

1.18 TWheelspeedDetail Struct Reference

`#include <wheel.h>`

Public Attributes

- EWheelId id
- float speedOfWheel

1.18.1 Detailed Description

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

1.18.2 Member Data Documentation

1.18.2.1 EWheelId TWheelspeedDetail::id

1.18.2.2 float TWheelspeedDetail::speedOfWheel

[m/s]

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

- wheel.h

1.19 TWheeltickDetail Struct Reference

`#include <wheel.h>`

Public Attributes

- EWheelId wheeltickIdentifier
- uint32_t wheeltickCounter

1.19.1 Detailed Description

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

1.19.2 Member Data Documentation

1.19.2.1 uint32_t TWheeltickDetail::wheeltickCounter

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

1.19.2.2 EWheelId TWheeltickDetail::wheeltickIdentifier

Type for which this tick information is provided.

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

- wheel.h

1.20 TWheelticks Struct Reference

`#include <wheel.h>`

Public Attributes

- uint64_t timestamp
- TWheeltickDetail elements [WHEEL_NUM_ELEMENTS]

1.20.1 Detailed Description

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

1.20.2 Member Data Documentation

1.20.2.1 TWheeltickDetail TWheelticks::elements[WHEEL_NUM_ELEMENTS]

The array of wheeltick elements.

1.20.2.2 uint64_t TWheelticks::timestamp

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

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

- wheel.h

2 File Documentation

2.1 acceleration.h File Reference

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

Classes

- struct TAccelerationConfiguration
- struct TAccelerationData

Typedefs

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

Enumerations

- enum EAccelerationConfigValidityBits { ACCELERATION_CONFIG_DISTX_V-ALID = 0x00000001, ACCELERATION_CONFIG_DISTY_VALID = 0x00000002, ACCELERATION_CONFIG_DISTZ_VALID = 0x00000004, ACCELERATIO-N_CONFIG_ANGLEYAW_VALID = 0x00000008, ACCELERATION_CONFI-G_ANGLEPITCH_VALID = 0x00000010, ACCELERATION_CONFIG_ANGL-EROLL_VALID = 0x00000020, ACCELERATION_CONFIG_SIGMAX_VALID = 0x00000040, ACCELERATION_CONFIG_SIGMAY_VALID = 0x00000080, ACCELERATION_CONFIG_SIGMAZ_VALID = 0x00000100, ACCELERATION-_CONFIG_TYPE_VALID = 0x00000200 }
- enum EAccelerationTypeBits { ACCELERATION_X_PROVIDED = 0x00000001, ACCELERATION_Y_PROVIDED = 0x00000002, ACCELERATION_Z_PR-OVIDED = 0x00000004, ACCELERATION_TEMPERATURE_PROVIDED = 0x00000008 }
- enum EAccelerationValidityBits { ACCELERATION_X_VALID = 0x00000001, ACCELERATION_Y_VALID = 0x00000002, ACCELERATION_Z_VALID = 0x00000004, ACCELERATION_TEMPERATURE_VALID = 0x00000008 }

Functions

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

2.1.1 Typedef Documentation

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

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

Parameters

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::anglePitch.

ACCELERATION_CONFIG_ANGLEROLL_VALID Validity bit for field TAccelerationConfiguration::angleRoll.

ACCELERATION_CONFIG_SIGMAX_VALID Validity bit for field TAccelerationConfiguration::sigmaX.

ACCELERATION_CONFIG_SIGMAY_VALID Validity bit for field TAccelerationConfiguration::sigmaX.

ACCELERATION_CONFIG_SIGMAZ_VALID Validity bit for field TAccelerationConfiguration::sigmaZ.

ACCELERATION_CONFIG_TYPE_VALID Validity bit for field TAcceleration-Configuration::typeBits.

2.1.2.2 enum EAccelerationTypeBits

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

Enumerator:

ACCELERATION_X_PROVIDED An acceleration measurement for the x-axis is provided

ACCELERATION_Y_PROVIDED An acceleration measurement for the y-axis is provided

ACCELERATION_Z_PROVIDED An acceleration measurement for the z-axis is provided

ACCELERATION_TEMPERATURE_PROVIDED A measurement for the temperature is provided

2.1.2.3 enum EAccelerationValidityBits

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

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

Enumerator:

ACCELERATION_X_VALID Validity bit for field TAccelerationData::x.

ACCELERATION_Y_VALID Validity bit for field TAccelerationData::y.

ACCELERATION_Z_VALID Validity bit for field TAccelerationData::z.

ACCELERATION_TEMPERATURE_VALID Validity bit for field TAcceleration-Data::temperature.

2.1.3 Function Documentation

2.1.3.1 bool snsAccelerationDeregisterCallback (AccelerationCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.1.3.2 bool snsAccelerationDestroy ( )

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

Returns

True if shutdown has been successfull.

2.1.3.3 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

2.1.3.4 bool snsAccelerationGetAccelerationData (TAccelerationData * *accelerationData*)

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

Parameters

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

Returns

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

2.1.3.5 bool snsAccelerationGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.1.3.6 bool snsAccelerationInit ( )

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

Returns

True if initialization has been successfull.

2.1.3.7 bool snsAccelerationRegisterCallback (AccelerationCallback *callback*)

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

Parameters

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 <stdbool.h>
```

Classes

- struct TGyroscopeConfiguration
- struct TGyroscopeData

Typedefs

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

Enumerations

- enum EGyroscopeConfigValidityBits { GYROSCOPE_CONFIG_ANGLEYAW_VALID = 0x00000001, GYROSCOPE_CONFIG_ANGLEPITCH_VALID = 0x00000002, GYROSCOPE_CONFIG_ANGLEROLL_VALID = 0x00000004,

GYROSCOPE_CONFIG_MOMENTYAW_VALID = 0x00000008, GYROSCOPE_CONFIG_SIGMAGYROSCOPE_VALID = 0x00000010, GYROSCOPE_CONFIG_TYPE_VALID = 0x00000020 }

- enum EGyroscopeTypeBits { GYROSCOPE_TEMPERATURE_COMPENSATED = 0x00000001, GYROSCOPE_YAWRATE_PROVIDED = 0x00000002, GYROSCOPE_PITCHRATE_PROVIDED = 0x00000004, GYROSCOPE_ROLLRATE_PROVIDED = 0x00000008, GYROSCOPE_TEMPERATURE_PROVIDED = 0x00000010 }
- enum EGyroscopeValidityBits { GYROSCOPE_YAWRATE_VALID = 0x00000001, GYROSCOPE_PITCHRATE_VALID = 0x00000002, GYROSCOPE_ROLLRATE_VALID = 0x00000004, GYROSCOPE_TEMPERATURE_VALID = 0x00000008 }

Functions

- bool snsGyroscopeInit ()
- bool snsGyroscopeDestroy ()
- bool snsGyroscopeGetMetaData (TSensorMetaData *data)
- bool snsGyroscopeGetConfiguration (TGyroscopeConfiguration *gyroConfig)
- bool snsGyroscopeGetGyroscopeData (TGyroscopeData *gyroData)
- bool snsGyroscopeRegisterCallback (GyroscopeCallback callback)
- bool snsGyroscopeDeregisterCallback (GyroscopeCallback callback)

2.2.1 Typedef Documentation

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

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

Parameters

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 EGyroscopeConfig-

ValidityBits values.

Enumerator:

GYROSCOPE_CONFIG_ANGLEYAW_VALID Validity bit for field TGyroscopeConfiguration::angleYaw.

GYROSCOPE_CONFIG_ANGLEPITCH_VALID Validity bit for field TGyroscopeConfiguration::anglePitch.

GYROSCOPE_CONFIG_ANGLEROLL_VALID Validity bit for field TGyroscopeConfiguration::angleRoll.

GYROSCOPE_CONFIG_MOMENTYAW_VALID Validity bit for field TGyroscopeConfiguration::momentOfYawInertia.

GYROSCOPE_CONFIG_SIGMAGYROSCOPE_VALID Validity bit for field TGyroscopeConfiguration::sigmaGyroscope.

GYROSCOPE_CONFIG_TYPE_VALID Validity bit for field TGyroscopeConfiguration::typeBits.

2.2.2.2 enum EGyroscopeTypeBits

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

Enumerator:

GYROSCOPE_TEMPERATURE_COMPENSATED Temperature bias compensation already applied to gyroscope signals.

GYROSCOPE_YAWRATE_PROVIDED A measurement for the z/yaw-axis is provided

GYROSCOPE_PITCHRATE_PROVIDED A measurement for the y/pitch-axis is provided

GYROSCOPE_ROLLRATE_PROVIDED A measurement for the x/roll-axis is provided

GYROSCOPE_TEMPERATURE_PROVIDED A measurement for the temperature is provided

2.2.2.3 enum EGyroscopeValidityBits

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

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

Enumerator:

GYROSCOPE_YAWRATE_VALID Validity bit for field TGyroscopeData::yawRate.

GYROSCOPE_PITCHRATE_VALID Validity bit for field TGyroscopeData::pitchRate.

GYROSCOPE_ROLLRATE_VALID Validity bit for field TGyroscopeData::rollRate.

GYROSCOPE_TEMPERATURE_VALID Validity bit for field TGyroscopeData::temperature.

2.2.3 Function Documentation

2.2.3.1 bool snsGyroscopeDeregisterCallback (GyroscopeCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.2.3.2 bool snsGyroscopeDestroy ( )

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

Returns

True if shutdown has been successfull.

2.2.3.3 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

2.2.3.4 bool snsGyroscopeGetGyroscopeData (TGyroscopeData * *gyroData*)

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

Parameters

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

2.2.3.5 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.

2.2.3.6 bool snsGyroscopeInit ( )

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

Returns

True if initialization has been successfull.

2.2.3.7 bool snsGyroscopeRegisterCallback (GyroscopeCallback *callback*)

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

Parameters

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 <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_TRAVERSE_VALID = 0x00000002, INCLINATION_STATUS_VALID = 0x00000004 }

Functions

- bool snsInclinationInit ()
- bool snsInclinationDestroy ()
- bool snsInclinationGetMetaData (TSensorMetaData ∗data)
- bool snsInclinationGetInclinationData (TInclinationData ∗inclination)
- bool snsInclinationRegisterCallback (InclinationCallback callback)
- bool snsInclinationDeregisterCallback (InclinationCallback callback)

2.3.1 Typedef Documentation

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

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

Parameters

inclination-Data	pointer to an array of TInclinationData with size numElements
num-Elements,:	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 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.

2.3.3.3 bool snsInclinationGetInclinationData (TInclinationData * *inclination*)

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

Parameters

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

2.3.3.4 bool snsInclinationGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.3.3.5 bool snsInclinationInit ( )

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

Returns

True if initialization has been successfull.

2.3.3.6 bool snsInclinationRegisterCallback (InclinationCallback *callback*)

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

Parameters

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 <stdbool.h>`

Classes

- struct TOdometerData

Typedefs

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

Enumerations

- enum EOdometerValidityBits { ODOMETER_TRAVELLEDDISTANCE_VALID = 0x00000001 }

Functions

- bool snsOdometerInit ()
- bool snsOdometerDestroy ()
- bool snsOdometerGetMetaData (TSensorMetaData ∗data)
- bool snsOdometerGetOdometerData (TOdometerData ∗odometer)
- bool snsOdometerRegisterCallback (OdometerCallback callback)
- bool snsOdometerDeregisterCallback (OdometerCallback callback)

2.4.1 Typedef Documentation

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

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

Parameters

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 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.

2.4.3.3 bool snsOdometerGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.4.3.4 bool snsOdometerGetOdometerData (TOdometerData * *odometer*)

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

Parameters

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

2.4.3.5 bool snsOdometerInit ( )

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

Returns

True if initialization has been successfull.

2.4.3.6 bool snsOdometerRegisterCallback (OdometerCallback *callback*)

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

Parameters

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 <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)

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

reverse-GearData	pointer to an array of TReverseGearData with size numElements
num-Elements,:	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 TReverseGear-Data::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 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.

2.5.3.3 bool snsReverseGearGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.5.3.4 bool snsReverseGearGetReverseGearData (TReverseGearData * *reverseGear*)

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

Parameters

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

2.5.3.5 bool snsReverseGearInit ( )

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

Returns

True if initialization has been successfull.

2.5.3.6 bool snsReverseGearRegisterCallback (ReverseGearCallback *callback*)

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

Parameters

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 <stdbool.h>
```

Classes

- struct TSlipAngleData

Typedefs

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

Enumerations

- enum ESlipAngleValidityBits { SLIPANGLE_SLIPANGLE_VALID = 0x00000001 }

Functions

- bool snsSlipAngleInit ()
- bool snsSlipAngleDestroy ()
- bool snsSlipAngleGetMetaData (TSensorMetaData ∗data)
- bool snsSlipAngleGetSlipAngleData (TSlipAngleData ∗data)
- bool snsSlipAngleRegisterCallback (SlipAngleCallback callback)
- bool snsSlipAngleDeregisterCallback (SlipAngleCallback callback)

2.6.1 Typedef Documentation

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

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

Parameters

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 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.

2.6.3.3 bool snsSlipAngleGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.6.3.4 bool snsSlipAngleGetSlipAngleData (TSlipAngleData * *data*)

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

Parameters

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

2.6.3.5 bool snsSlipAngleInit ( )

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

Returns

True if initialization has been successfull.

2.6.3.6 bool snsSlipAngleRegisterCallback (SlipAngleCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.7 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)

2.7.1 Enumeration Type Documentation

2.7.1.1 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.

2.7.1.2 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

2.7.2 Function Documentation

2.7.2.1 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

2.8 sns.h File Reference

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

Defines

- #define GENIVI_SNS_API_MAJOR 2
- #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)

2.8.1 Define Documentation

2.8.1.1 #define GENIVI_SNS_API_MAJOR 2

2.8.1.2 #define GENIVI_SNS_API_MICRO 0

2.8.1.3 #define GENIVI_SNS_API_MINOR 0

2.8.2 Function Documentation

2.8.2.1 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.

2.8.2.2 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.

2.8.2.3 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.

2.9 steering-angle.h File Reference

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

Classes

- struct TSteeringAngleData
- struct TSteeringAngleConfiguration

Typedefs

- typedef void(* SteeringAngleCallback)(const TSteeringAngleData steeringAngleData[], uint16_t numElements)

Enumerations

- enum ESteeringAngleValidityBits { STEERINGANGLE_FRONT_VALID = 0x00000001, STEERINGANGLE_REAR_VALID = 0x00000002, STEERINGANGLE_STEERINGWHEEL_VALID = 0x00000004 }

Functions

- bool snsSteeringAngleInit ()
- bool snsSteeringAngleDestroy ()
- bool snsSteeringAngleGetMetaData (TSensorMetaData *data)
- bool snsSteeringAngleGetSteeringAngleData (TSteeringAngleData *angleData)
- bool snsSteeringAngleGetConfiguration (TSteeringAngleConfiguration *config)
- bool snsSteeringAngleRegisterCallback (SteeringAngleCallback callback)
- bool snsSteeringAngleDeregisterCallback (SteeringAngleCallback callback)

2.9.1 Typedef Documentation

2.9.1.1 typedef void(* SteeringAngleCallback)(const TSteeringAngleData steeringAngleData[], uint16_t numElements)

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

Parameters

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

2.9.2 Enumeration Type Documentation

2.9.2.1 enum ESteeringAngleValidityBits

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

Enumerator:

STEERINGANGLE_FRONT_VALID Validity bit for field TSteeringAngleData::front.

STEERINGANGLE_REAR_VALID Validity bit for field TSteeringAngleData::rear.

STEERINGANGLE_STEERINGWHEEL_VALID Validity bit for field TSteeringAngleData::steeringWheel.

2.9.3 Function Documentation

2.9.3.1 bool snsSteeringAngleDeregisterCallback (SteeringAngleCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.9.3.2 bool snsSteeringAngleDestroy ( )

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

Returns

True if shutdown has been successfull.

2.9.3.3 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

2.9.3.4 bool snsSteeringAngleGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.9.3.5 bool snsSteeringAngleGetSteeringAngleData (TSteeringAngleData * *angleData*)

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

Parameters

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

2.9.3.6 bool snsSteeringAngleInit ( )

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

Returns

True if initialization has been successfull.

2.9.3.7 bool snsSteeringAngleRegisterCallback (SteeringAngleCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.10 vehicle-data.h File Reference

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

Classes

- struct TDistance3D

Enumerations

- enum EAxleType { SNS_AXLE_UNKNOWN = 0, SNS_AXLE_FRONT = 1, SNS_AXLE_REAR = 2, SNS_AXLE_BOTH = 3 }
- enum EVehicleType { SNS_CAR = 0, SNS_TRUCK = 1, SNS_MOTORBIKE = 2, SNS_BUS = 3 }

Functions

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

2.10.1 Enumeration Type Documentation

2.10.1.1 enum EAxleType

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

Enumerator:

SNS_AXLE_UNKNOWN It is not known which axles are driven.

SNS_AXLE_FRONT Only the front axle is driven.

SNS_AXLE_REAR Only the rear axle is driven.

SNS_AXLE_BOTH Both axles are driven (4 wheel drive).

2.10.1.2 enum EVehicleType

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

Enumerator:

SNS_CAR Passenger car with 4 wheels.

SNS_TRUCK Truck

SNS_MOTORBIKE Motorbike with 2 wheels.

SNS_BUS Passenger bus.

2.10.2 Function Documentation

2.10.2.1 bool vehicleDataGetDistCoG2RefPoint (TDistance3D * *dist*)

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

Parameters

dist	The distance result in 3 dimensions is written to this parameter.

Returns

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

2.10.2.2 bool vehicleDataGetDistFrontAxle2RefPoint (TDistance3D * *dist*)

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

Parameters

dist	The distance result in 3 dimensions is written to this parameter.

Returns

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

2.10.2.3 bool vehicleDataGetDistRearAxle2RefPoint (TDistance3D * *dist*)

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

Parameters

dist	The distance result in 3 dimensions is written to this parameter.

Returns

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

2.10.2.4 bool vehicleDataGetDrivenAxles (EAxleType * *axles*)

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

Parameters

axles	The driven axles type is written to this parameter as a result as defined in the enumeration.

Returns

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

2.10.2.5 bool vehicleDataGetFrontAxleTrackWidth (float * *width*)

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

Parameters

width	The vehicle track width of the front axle in m is written to this parameter as a result.

Returns

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

2.10.2.6 bool vehicleDataGetSeatCount (uint8_t * *seatCount*)

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

Parameters

seatCount	The number of seats is written to this parameter as a result.

Returns

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

2.10.2.7 bool vehicleDataGetTrackWidth (float * *width*)

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

Parameters

width	The vehicle track width in m is written to this parameter as a result.

Returns

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

2.10.2.8 bool vehicleDataGetVehicleMass (float * *mass*)

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

Parameters

mass	The vehicle mass in kg is written to this parameter as a result.

Returns

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

2.10.2.9 bool vehicleDataGetVehicleType (EVehicleType * *type*)

Type of the vehicle. This is a static configuration.

Parameters

type	The vehicle type is written to this parameter as a result as defined in the enumeration.

Returns

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

2.10.2.10 bool vehicleDataGetVehicleWidth (float * *width*)

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

Parameters

width	The vehicle width in m is written to this parameter as a result.

Returns

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

2.10.2.11 bool vehicleDataGetWheelBase (float * *wheelbase*)

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

Parameters

width	The wheel base in m is written to this parameter as a result.

Returns

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

2.11 vehicle-speed.h File Reference

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

Classes

- struct TVehicleSpeedData

Typedefs

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

Enumerations

- enum EVehicleSpeedValidityBits { VEHICLESPEED_VEHICLESPEED_VALID = 0x00000001 }

Functions

- bool snsVehicleSpeedInit ()
- bool snsVehicleSpeedDestroy ()
- bool snsVehicleSpeedGetMetaData (TSensorMetaData ∗data)
- bool snsVehicleSpeedGetVehicleSpeedData (TVehicleSpeedData ∗vehicleSpeed)
- bool snsVehicleSpeedRegisterCallback (VehicleSpeedCallback callback)
- bool snsVehicleSpeedDeregisterCallback (VehicleSpeedCallback callback)

2.11.1 Typedef Documentation

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

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

Parameters

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

2.11.2 Enumeration Type Documentation

2.11.2.1 enum EVehicleSpeedValidityBits

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

Enumerator:

VEHICLESPEED_VEHICLESPEED_VALID Validity bit for field TVehicleSpeedData::vehicleSpeed.

2.11.3 Function Documentation

2.11.3.1 bool snsVehicleSpeedDeregisterCallback (VehicleSpeedCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.11.3.2 bool snsVehicleSpeedDestroy ( )

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

Returns

True if shutdown has been successfull.

2.11.3.3 bool snsVehicleSpeedGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.11.3.4 bool snsVehicleSpeedGetVehicleSpeedData (TVehicleSpeedData * *vehicleSpeed*)

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

Parameters

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

Returns

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

2.11.3.5 bool snsVehicleSpeedInit ( )

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

Returns

True if initialization has been successfull.

2.11.3.6 bool snsVehicleSpeedRegisterCallback (VehicleSpeedCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.12 vehicle-state.h File Reference

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

Classes

- struct TVehicleStateData

Typedefs

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

Enumerations

- enum EVehicleStateValidityBits { VEHICLESTATE_ANTILOCKBRAKESYSTEMACTIVE_VALID = 0x00000001, VEHICLESTATE_BRAKEACTIVE_VALID = 0x00000002, VEHICLESTATE_ELECTRONICSTABILITYPROGRAMACTIVE_VALID = 0x00000004, VEHICLESTATE_TRACTIONCONTROLACTIVE_VALID = 0x00000008 }

Functions

- bool snsVehicleStateInit ()
- bool snsVehicleStateDestroy ()
- bool snsVehicleStateGetMetaData (TSensorMetaData *data)
- bool snsVehicleStateGetVehicleStateData (TVehicleStateData *vehicleState)
- bool snsVehicleStateRegisterCallback (VehicleStateCallback callback)
- bool snsVehicleStateDeregisterCallback (VehicleStateCallback callback)

2.12.1 Typedef Documentation

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

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

Parameters

vehicleSpeedData	pointer to an array of TVehicleStateData 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 EVehicleStateValidityBits

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

Enumerator:

VEHICLESTATE_ANTILOCKBRAKESYSTEMACTIVE_VALID Validity bit for field antiLockBrakeSystemActive::antiLockBrakeSystemActive.

VEHICLESTATE_BRAKEACTIVE_VALID Validity bit for field antiLockBrake-

SystemActive::brakeActive.

VEHICLESTATE_ELECTRONICSTABILITYPROGRAMACTIVE_VALID Validity bit for field antiLockBrakeSystemActive::electronicStabilityProgramActive.

VEHICLESTATE_TRACTIONCONTROLACTIVE_VALID Validity bit for field antiLockBrakeSystemActive::tractionControlActive.

2.12.3 Function Documentation

2.12.3.1 bool snsVehicleStateDeregisterCallback (VehicleStateCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.12.3.2 bool snsVehicleStateDestroy ( )

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

Returns

True if shutdown has been successfull.

2.12.3.3 bool snsVehicleStateGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.12.3.4 bool snsVehicleStateGetVehicleStateData (TVehicleStateData * *vehicleState*)

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

Parameters

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

2.12.3.5 bool snsVehicleStateInit ( )

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

Returns

True if initialization has been successfull.

2.12.3.6 bool snsVehicleStateRegisterCallback (VehicleStateCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.13 wheel.h File Reference

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

Classes

- struct TWheeltickDetail
- struct TWheelticks
- struct TWheelspeedAngularDetail
- struct TWheelspeedAngular
- struct TWheelspeedDetail
- struct TWheelspeed

Defines

- #define WHEEL_NUM_ELEMENTS 4

Typedefs

- typedef void(∗ WheeltickCallback)(const TWheelticks ticks[], uint16_t numElements)
- typedef void(∗ WheelspeedAngularCallback)(const TWheelspeedAngular rotations[], uint16_t numElements)
- typedef void(∗ WheelspeedCallback)(const TWheelspeed wheelspeed[], uint16_t numElements)

Enumerations

- enum EWheelId { WHEEL_INVALID = 0, WHEEL_UNKNOWN = 1, WHEEL_AXLE_NONDRIVEN = 2, WHEEL_AXLE_FRONT = 3, WHEEL_AXLE_REAR = 4, WHEEL_LEFT_FRONT = 5, WHEEL_RIGHT_FRONT = 6, WHEEL_LEFT_REAR = 7, WHEEL_RIGHT_REAR = 8 }

Functions

- bool snsWheeltickInit ()
- bool snsWheeltickDestroy ()
- bool snsWheelTickGetMetaData (TSensorMetaData ∗data)
- bool snsWheeltickGetWheelticksPerRevolution (uint16_t ∗ticks)
- bool snsWheeltickGetWheelticksCountMax (uint32_t ∗max_count)
- bool snsWheeltickGetWheelticks (TWheelticks ∗ticks)
- bool snsWheeltickRegisterCallback (WheeltickCallback callback)
- bool snsWheeltickDeregisterCallback (WheeltickCallback callback)
- bool snsWheelspeedAngularInit ()
- bool snsWheelspeedAngularDestroy ()
- bool snsWheelspeedAngularGetMetaData (TSensorMetaData ∗data)
- bool snsWheelspeedAngularGet (TWheelspeedAngular ∗wsa)
- bool snsWheelspeedAngularRegisterCallback (WheelspeedAngularCallback callback)
- bool snsWheelspeedAngularDeregisterCallback (WheelspeedAngularCallback callback)
- bool snsWheelspeedInit ()
- bool snsWheelspeedDestroy ()
- bool snsWheelspeedGetMetaData (TSensorMetaData ∗data)
- bool snsWheelspeedGet (TWheelspeed ∗wheelspeed)
- bool snsWheelspeedRegisterCallback (WheelspeedCallback callback)
- bool snsWheelspeedDeregisterCallback (WheelspeedCallback callback)
- bool snsWheelGetTireRollingCircumference (float ∗circumference)

2.13.1 Define Documentation

2.13.1.1 #define WHEEL_NUM_ELEMENTS 4

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

2.13.2 Typedef Documentation

2.13.2.1 typedef void(* WheelspeedAngularCallback)(const TWheelspeedAngular rotations[], uint16_t numElements)

Callback type for angular wheel speed sensor service. Use this type of callback if you want to register for angular wheel speed data. Wheel tick data can be provided for multiple WheelIds. E.g. data can be provided for all 4 wheels. This callback may return buffered data (numElements >1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (numElements >1), the elements will be ordered with rising timestamps

Parameters

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

2.13.2.2 typedef void(* WheelspeedCallback)(const TWheelspeed wheelspeed[], uint16_t numElements)

Callback type for wheel speed sensor service. Use this type of callback if you want to register for wheel speed data. Wheel tick data can be provided for multiple WheelIds. E.g. data can be provided for all 4 wheels. This callback may return buffered data (numElements >1) for different reasons for (large) portions of data buffered at startup for data buffered during regular operation e.g. for performance optimization (reduction of callback invocation frequency) If the array contains (numElements >1), the elements will be ordered with rising timestamps

Parameters

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

2.13.2.3 typedef void(* WheeltickCallback)(const TWheelticks ticks[], uint16_t numElements)

Callback type for wheel tick sensor service. Use this type of callback if you want to register for wheel tick data. Wheel tick data can be provided for multiple WheelIds. E.g. data can be provided for all 4 wheels. This callback may return buffered data (num-

Elements >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

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

2.13.3 Enumeration Type Documentation

2.13.3.1 enum EWheelId

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

Enumerator:

WHEEL_INVALID Wheel tick data is invalid / the field is unused.

WHEEL_UNKNOWN No information available where the wheel tick data is from.

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

WHEEL_AXLE_FRONT Wheel tick data is an average from the front axle.

WHEEL_AXLE_REAR Wheel tick data is an average from the rear axle.

WHEEL_LEFT_FRONT Wheel tick data is from the left front wheel.

WHEEL_RIGHT_FRONT Wheel tick data is from the right front wheel.

WHEEL_LEFT_REAR Wheel tick data is from the left rear wheel.

WHEEL_RIGHT_REAR Wheel tick data is from the right rear wheel.

2.13.4 Function Documentation

2.13.4.1 bool snsWheelGetTireRollingCircumference (float * *circumference*)

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

Parameters

circumference	The tire rolling circumference in m is written to this parameter as a result.

Returns

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

2.13.4.2 bool snsWheelspeedAngularDeregisterCallback (WheelspeedAngularCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.13.4.3 bool snsWheelspeedAngularDestroy ( )

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

Returns

True if shutdown has been successfull.

2.13.4.4 bool snsWheelspeedAngularGet (TWheelspeedAngular * *wsa*)

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

Parameters

wsa	After calling the method the currently available angular wheel speed data is written into the array.

Returns

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

2.13.4.5 bool snsWheelspeedAngularGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.13.4.6 bool snsWheelspeedAngularInit ( )

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

Returns

True if initialization has been successfull.

2.13.4.7 bool snsWheelspeedAngularRegisterCallback (WheelspeedAngularCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.13.4.8 bool snsWheelspeedDeregisterCallback (WheelspeedCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.13.4.9 bool snsWheelspeedDestroy ( )

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

Returns

True if shutdown has been successfull.

2.13.4.10 bool snsWheelspeedGet (TWheelspeed * *wheelspeed*)

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

Parameters

wheelspeed	After calling the method the currently available wheel speed data is written into the array.

Returns

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

2.13.4.11 bool snsWheelspeedGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.13.4.12 bool snsWheelspeedInit ( )

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

Returns

True if initialization has been successfull.

2.13.4.13 bool snsWheelspeedRegisterCallback (WheelspeedCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.

2.13.4.14 bool snsWheeltickDeregisterCallback (WheeltickCallback *callback*)

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

Parameters

callback	The callback which should be deregistered.

Returns

True if callback has been deregistered successfully.

2.13.4.15 bool snsWheeltickDestroy ( )

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

Returns

True if shutdown has been successfull.

2.13.4.16 bool snsWheelTickGetMetaData (TSensorMetaData * *data*)

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

Parameters

data	Meta data content about the sensor service.

Returns

True if meta data is available.

2.13.4.17 bool snsWheeltickGetWheelticks (TWheelticks * *ticks*)

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

Parameters

ticks	After calling the method the currently available wheel tick data is written into the array.

Returns

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

2.13.4.18 bool snsWheeltickGetWheelticksCountMax (uint32_t * *max_count*)

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

Parameters

max_count	The maximum value of wheel tick rolling counters

Returns

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

2.13.4.19 bool snsWheeltickGetWheelticksPerRevolution (uint16_t * *ticks*)

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

Parameters

ticks	The number of ticks for a single revolution of one wheel

Returns

True if configuration data is available.

2.13.4.20 bool snsWheeltickInit ( )

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

Returns

True if initialization has been successfull.

2.13.4.21 bool snsWheeltickRegisterCallback (WheeltickCallback *callback*)

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

Parameters

callback	The callback which should be registered.

Returns

True if callback has been registered successfully.