SENSORIS Interface Architecture

SENSORIS Innovation Platform hosted by ERTICO - ITS Europe

Version 1.4.0

Table of Contents

1. Introduction

The **Sensor Interface Specification (SENSORIS)** defines an interface for requesting and sending vehicle sensor data from vehicles to clouds and across clouds. The specification and its standardization focus on the content and encoding of the interface.

SENSORIS differentiates between three **actor roles**, which are shown in Figure 1. A vehicle is part of a **vehicle fleet**. The vehicles of a vehicle fleet communicate with a **vehicle cloud**. A vehicle cloud can also communicate with a **service cloud**. Vehicle fleet, vehicle cloud, and service cloud are actor roles. A cloud instance can have both the role of a vehicle cloud and a service cloud. However, if a cloud instance has only the role of a service cloud, then it cannot communicate with a vehicle fleet. An example setup could be that vehicles of an OEM vehicle fleet communicate with their OEM vehicle cloud. The OEM vehicle cloud in turn communicates also to the service cloud of a map maker.

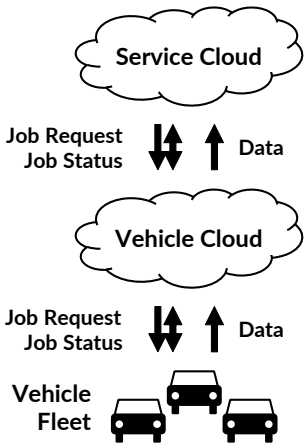


Figure 1. Actor roles and interface

The **interface** of SENSORIS defines content and encoding of the messages that are communicated between the actor roles. **Data messages** contain vehicle sensor data. Data messages communicated from one vehicle of a vehicle fleet to its vehicle cloud contain sensor data from the one vehicle. Data messages communicated from a vehicle cloud to a service cloud contain data from individual vehicles or aggregated data from several vehicles of a vehicle fleet. **Job request messages** contain jobs defining which vehicle sensor data is requested under which conditions and when the data shall be communicated to the requesting cloud. **Job status messages** contain information about termination of jobs. Job status messages communicated from a vehicle of a vehicle fleet to its vehicle cloud or from a vehicle cloud to a service cloud contain the reason of the termination of the job in the vehicle or vehicle cloud. Job status messages communicated from a service cloud to a vehicle cloud or from a vehicle cloud to a vehicle of a vehicle fleet request the termination of the job.

SENSORIS is not limited to just one instance of each actor role as shown in Figure 1, but is designed for **cross-collaboration in a setup with multiple actor roles** as shown in Figure 2. A vehicle cloud can communicate with an arbitrary number of vehicle fleets. A service cloud can communicate with other service and vehicle clouds. For all communication channels the interface contains job request, job status, and data message types.

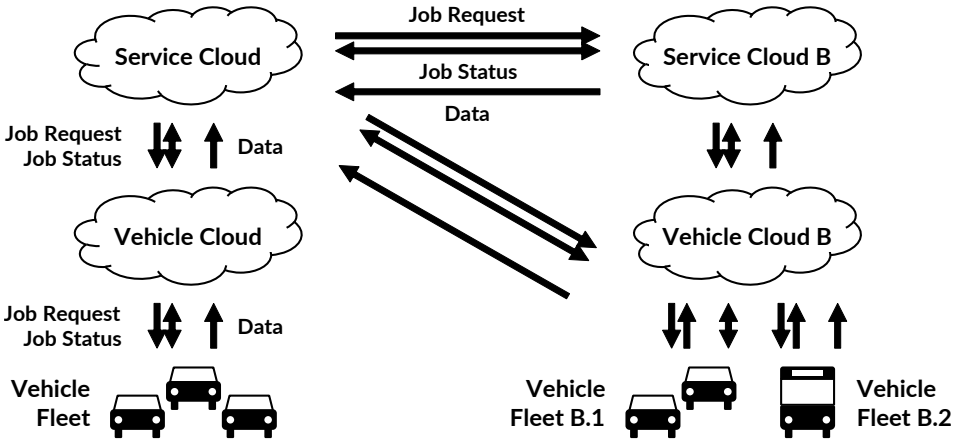


Figure 2. Multiple actor roles and interface

The SENSORIS interface covers a wide range of vehicle sensors from **Standard Definition (SD), over High Definition (HD), to Automated Driving (AD)** as shown in Figure 3. The large variety in quantity and quality of sensors enable use and reuse of vehicle sensor data for a multitude of use cases.

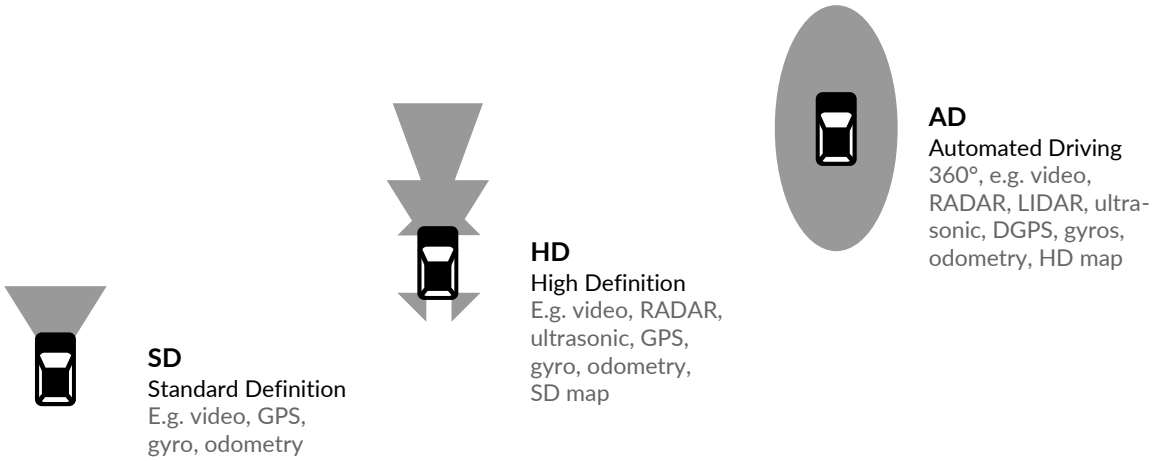


Figure 3. Range of vehicle sensors from SD, over HD, to AD

This document defines the interface architecture of SENSORIS. Aspects that shall be taken into account for defining the reference implementation architecture are also described in this document.

The architecture definition is structured as follows. In Chapter 2 the interface architecture view on content and encoding of SENSORIS messages is described. The document concludes with a Glossary.

2. Interface Architecture

The **interface architecture is purely limited to content and encoding of the SENSORIS interface**. This limitation serves two purposes. The first purpose is to allow for a large variety of implementations. The SENSORIS interface shall be e.g. irrespective of the communication channel used, may it be already available technology to retrofit vehicle fleets being already in the field, state-of-the-art technology to roll out on vehicle fleets in production, or next generation technology for research. The second purpose of the limitation is to reduce time to standardization, as requirements for implementation sometimes differ significantly.

The sections in this chapter build on each other. Standardized reference systems used in SENSORIS are described in Section 2.1. SENSORIS message encoding in Section 2.2 details the binary encoding and its cross-platform compatibility. Conventions concerning versioning and naming are listed in Section 2.3. Section 2.4, Section 2.5, and Section 2.6 describe the schema skeleton based on interface requirements for data messages, job request messages, and job status messages respectively.

2.1. Reference Systems

SENSORIS uses **standardized reference systems**, namely International System of Units, Coordinated Universal Time, World Geodetic System 1984, and reference frames in road vehicle dynamics.

The **International System of Units (SI)** is the most important system of units of measurement. It is defined in the SI Brochure, which is published by the Bureau International des Poids et Mesures (BIPM).[1] A product of a number and a unit expresses the value of a quantity. The SI base quantities and base units used in SENSORIS are listed in Table 1. Derived SI units are products of powers of base units. The derived SI units used in SENSORIS are also listed in Table 1. Besides the SI units also non-SI units are accepted for use with SI. These non-SI units used in SENSORIS are also listed in Table 1.

Type	Quantity name	Unit name	Unit symbol
Base	Length	Metre	m
	Mass	Kilogram	kg
	Time, duration	Second	s
	Electric current	Ampere	A
Derived	Frequency	Hertz	Hz
	Pressure, stress	Pascal	Pa
	Power, radiant flux	Watt	W
	Electric charge, amount of electricity	coulomb	C
	Electrical potential difference, electromotive force	volt	V
	Celsius temperature	degree Celsius	°C
Non-SI	Time	minute	min
		hour	h
		day	d
	Plane angle	degree	°
	Volume	litre	l

Table 1. SI base/SI derived/non-SI quantities and units used in SENSORIS

SI also defines prefix names and prefix symbols that express decimal multiples and submultiples for SI units.

The prefixes used in SENSORIS are listed in Table 2.

Factor	Name	Symbol
10^{-3}	milli	m
10^{-6}	micro	µ
10^{3}	kilo	k

Table 2. SI prefix names and prefix symbols used in SENSORIS

SI also states that the value of dimensionless quantities may be expressed by the symbol % (percent) to represent the number 0.01.

Table 1 and Table 2 should not be considered complete.

The **Coordinated Universal Time (UTC)** is the most important time standard. Neither time zones nor daylight savings time are considered for UTC. Leap seconds are inserted at irregular intervals to keep UTC aligned to rotation of Earth. SENSORIS is based on UTC as reference, but omits leap seconds.

The **World Geodetic System 1984 (WGS84)** is the best known geodetic reference system and is used in cartography, geodesy, and navigation. The WGS84 standard defines coordinate system, reference ellipsoid, and geoid for positions on Earth. Positions are commonly expressed in a geographic coordinate system as longitude (east/west) and latitude (south/north) in degree for the horizontal position and as altitude or elevation (height) in metre for the vertical position. Longitude is zero degrees at the International Reference Meridian, which is located near the Greenwich meridian. Latitude is zero degrees at the equator. Altitude is zero meters on the WGS84 reference ellipsoid. Elevation is derived from altitude and refers to the Earth Gravitational Model 1996 (EGM96) as geoid, which defines the nominal sea level surface. Positions in SENSORIS are given as longitude, latitude, and altitude, i.e. giving the height as elevation is explicitly not allowed. This allows for a consistent derivation of elevation values in the cloud irrespective of the actual derivation in the respective vehicle sensors.

The **International Organization for Standardization (ISO) standard 8855:20112** defines the principal terms used for road vehicle dynamics.[2] These terms are used to specify the **SENSORIS vehicle coordinate system**. A vehicle coordinate system is defined by the ISO standard as combination of vehicle reference point and vehicle axis system. The vehicle reference point is the origin of the vehicle coordinate system. In SENSORIS the vehicle reference point is located by convention in the middle of the rear axis. The vehicle axis system of SENSORIS is right-handed and has a longitudinal x, lateral y, and vertical z axis. The longitudinal x axis points horizontally to the front of the vehicle. The lateral y axis points to the left of the vehicle. The vertical z axis points opposite to the gravitation vector to the top of the vehicle. The x, y, and z axis form an orthogonal axis system. The rotation around the longitudinal x axis is defined by the roll angle, around the lateral y axis by the pitch angle, and around the vertical z axis by the yaw angle. With the exception of GNSS sensor data, all measurements of sensors are transformed from their sensor coordinate system to the SENSORIS vehicle coordinate system based on proper calibration of the sensors. The reference point of the GNSS sensor data is given as a 3D translation to the SENSORIS vehicle reference point. For vehicles with trailers only the towing vehicle is considered in the SENSORIS vehicle coordinate system.

In addition to the SENSORIS vehicle coordinate system the length, width, and height of the vehicle are specified with a total of six values in positive and negative direction of the x, y, and z axis respectively. These measurements enable derivation of relative distances to the vehicle frame, e.g. of the relative distance

between an obstacle and the vehicle front bumper from the absolute distance between the vehicle reference point and the same obstacle.

2.2. Message Encoding

SENSORIS job request, job status, and data messages are communicated between the three actor roles vehicle fleet, vehicle cloud, and service cloud. The SENSORIS messages have to be **encoded for over-the-air and over-the-wire communication channels**, i.e. they have to be serialized by the sender prior to communication and then have to be deserialized by the receiver.

Message encoding has to fulfil several requirements. Over-the-air communication channels are normally limited in bandwidth and communication costs have to be considered. A small difference in data size over a fleet of several million vehicles sums up quickly to a large difference in overall data size. Therefore **the size of serialized data shall be minimized** by choosing a compact data serialization format. The two environments vehicle and cloud differ significantly regarding resources, operating systems, and programming languages used for software implementation. Resources in a vehicle are limited and expensive, namely processor performance and size of memory. Programs are usually implemented in C/C++ and run on top of a UNIX based operating system. In contrast, resources in a cloud environment are easily available. A UNIX based operating system is also often used in clouds. Programs are implemented in a variety of languages, e.g. Java, Python, JavaScript, and C++. The differences between the vehicle and cloud environments result in the requirement that **data serialization shall be able to cope with a variety of resource sets, operating systems, and programming languages**. For keeping licencing fees for message encoding at zero cost, data serialization shall use a library with a permissive license, e.g. Apache Licence version 2.

Additional requirements regarding message content also have to be fulfilled by SENSORIS message encoding. **Encoding shall support evolution of the data format**, i.e. adding new data types or fields shall be backward compatible so that the new data format can be read by both new code and code generated for previous versions of the data format. For textual data types it shall be ensured that **internationalization** is covered by the data serialization format, i.e. text in different languages can be encoded. The encoder also shall **support null values**, i.e. it shall be possible to explicitly not set a field value. Finally, the encoding shall **allow for proprietary extension of the data format**, e.g. for prototyping or research purposes.

A large variety of data serialization formats is currently available.[3] Candidates for SENSORIS fulfilling the listed requirements are Apache Avro, Apache Thrift, and Google Protocol Buffers. All of them provide an Interface Description Language (IDL) in which the schema can be strictly typed. Data serialization results in a compact byte array. The two Apache Projects have an Apache Licence version 2, whereas Google Protocol Buffers has a BSD 3-Clause licence. All of them support evolution of the data format, internationalization, null values, and the possibility to define propriety extensions. Google Protocol Buffers is used for SENSORIS as it is the most commonly used of the three data formats and also has been used for the Vehicle Sensor Data Cloud Ingestion Interface Specification published in 2015 by HERE.

Google Protocol Buffers (protobuf), are a language-neutral, platform-neutral, and extensible mechanism for serializing structured data.[4] For SENSORIS, version 3 of the protobuf library is used, which adds a streamlined approach for proprietary extensions. Using protobuf starts with defining a data schema as protobuf message types, see Figure 4. Then the protobuf compiler is run with the data schema as input and generates data access classes in one of the supported languages C++, Java, Python, Go, Ruby, C#, Objective C, JavaScript, or PHP. The compiler is only run initially and on schema changes.

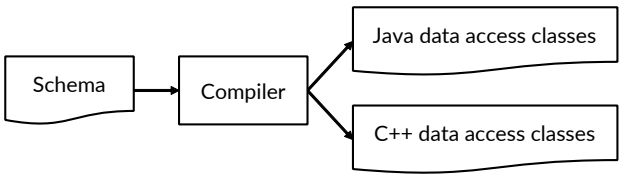


Figure 4. Protobuf schema, compiler, and auto-generated data classes

Usage of the auto-generated protobuf data access classes in the context of message encoding is shown in Figure 5. The communication from a vehicle of a vehicle fleet to its vehicle cloud is used as example in the following. On the vehicle the obtained sensor data is filled into the C++ data access classes. The class instances are then serialized into a byte array by the also auto-generated C++ encoder. The serialized data is transferred over-the-air to the vehicle cloud. There the auto-generated Java decoder deserializes the byte array into Java class instances having the same schema and sensor data as the C++ class instances on the vehicle.

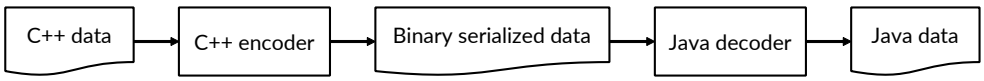


Figure 5. Example protobuf workflow for message encoding

In SENSORIS the protobuf scalar value types int64, bool, string, and bytes are used. String supports UTF8 encoded text, which fulfils the requirement for internationalization. Unsigned integers are not used in SENSORIS, as protobuf represents unsigned integers in Java by their signed counterparts, which may end up in confusion during use of the data classes. Protobuf enumerations, nested types, and imports from other schema files are also used in SENSORIS. The OneOf mechanism, which allows for a protobuf message with several fields where at most one field is set at the same time, is also used. The protobuf Any message type fulfils the requirement for proprietary extensions.[5] For a proprietary extension first the protobuf schema has to be defined. Then data can be encoded as a byte array of type bytes using the extension format. Finally, an Any message can be built which contains the encoded data and a type URL which acts as a globally unique identifier for the proprietary extension. On deserialization protobuf reads the URL and then is able to unpack the data from the byte array.

The requirement for null values is fulfilled by using protobuf **wrapper message types**.[6] These message types wrap a non-nullable scalar value type, e.g. string, in a message type which is nullable.

Multiplier usage

For the multiplier usage the following glossary is defined:

- BaseType: The declaration of a message within the protobuf schema (e.g. message EventGroup {...}). Similar to a class declaration in Java.
- Attribute field: the protobuf attribute declaration within a message of a primitive or complex datatype.
- AbsolutePath: an array of field numbers that are defining the path through the message structure starting from the root baseType "DataMessages" by using the number of the attribute field within its message.
- AbsolutePathString: is the string representation of the path through the message structure defined by the attribute field names starting from the root ("DataMessages") and separated by ":". Absolute Path and PathString are interchangeable.
- Example:
  - AbsolutePathString:

        - BaseType: sensoris.protobuf.messages.data.DataMessages
        - "data_message:event_group:localization_category:vehicle_position_and_orientation:position_and_accuracy:metric_ecef"
    - AbsolutePath: [2,2,2,2,2,4] is equivalent to above AbsolutePathString given the same baseType.
    - baseType and attribute field: above AbsolutePathString and AbsolutePath point to the attribute_field "metric_ecef" of the base type "sensoris.protobuf.types.spatial.PositionAndAccuracy.Metric"

Within SENSORIS, scalar values are overall exclusively represented by integer values of data type int64 with an implicit exponent to the base of 10 as a factor realising a fixed decimal digit encoding. The encoding from a measured value to an encoded value is done as: "encodedValue = measuredValue * $10^{exponent}$"

The exponent is provided either implicit through the protobuf schema or through an explicit override within a message.

Implicit exponent declaration

The exponent may be defined for a single attribute_field of type int64 within a message, for which the exponent is valid for any representation of that message type.

Exponent usage with simple attributes:

- Protobuf: Int64 value = 1 [(exponent) = 3]
- Measurement: measured_value = 123.456789
- Encoded: **value = 123456**
- Interpretation: real_value = 123.456

The exponent may also be used for an attribute field of a message type (e.g. Int64Value, Int64ValueAndAccuracy, XyzVectorAndAccuracy…) for which the exponent is propagated towards the attribute fields within the lower levels of the data structure.

Examples:

Exponent usage with complex data types:

- Protobuf: Int64ValueAndAccuracy attribute_n = 1 [(exponent) = 4]
- Measurements: attribute_n.measured_value 123.456789 with attribute_n.measured_accuracy 0.01
- Encoded: **attribute_n.value 1234567** with **attribute_n.accuracy 100**
- Decoded: attribute_n.value 123.4567 with attribute_n.accuracy 0.01

In case of duplication, the lower level exponent (simple type) is overridden by the higher level exponent (message type). For readability purpose, the exponent is also described with the comment "@resolution". In case of conflict, the attribute option "exponent" is the reference value.

Explicit exponent declaration

Within one message, the implicit exponent can be overridden declaring one specific attribute to be encoded with a different exponent.

The message type FieldResolutionOverride specifies:

- A baseType, describing the message that shall be used as root message for the following NodeArray. If no messageCode is provided, "DataMessages" is used as root message. Only one messageCode per FieldResolutionOverride is allowed
- A NodeArray, describing the attribute with the root message described by the messageCode. A NodeArray is mandatory.
- An exponent describing the exponent number. If no exponent is provided, the value 0 is used. Exponent = 0 results in "measured_value == encoded_value"

Examples:

- Content of one FieldResolutionOverride-Object
  - messageCode: "sensoris.protobuf.types.spatial.PositionAndAccuracy.Metric"
  - nodeArray: [1]
  - exponent: 5
- Interpretation: The x-value (number 1) within any representation of the Metric PositionAndAccuracy content in the protobuf message is encoded using 5 digits of accuracy. The value has to be divided by 10^5 to obtain the real value.

Based on bilateral sender-receiver agreement, the transported data size may be reduced by any state of the art **compression** of the protobuf binary serialized data. Compression reduces the size of the encoded data, but only above a certain size threshold. This threshold can be determined by tests with typical payloads. Specification of a specific compression algorithm is out of scope of the SENSORIS architecture.

SENSORIS **timestamps** are based on **UNIX time**, which is also known as POSIX time. The SENSORIS timestamp is encoded as number of milliseconds and fractions of milliseconds with microsecond resolution since the date 1970-01-01T00:00:00Z UTC (see reference systems in Section 2.1). At a vehicle speed of 50 m/s, i.e. 180 km/h, the distance travelled within 1 millisecond is 5 centimetre. If a higher spatiotemporal accuracy is required, then also the microsecond fraction of the SENSORIS timestamp can be used. The timestamp is independent of time zones and daylights savings time. It assumes that all minutes are exactly 60 seconds long, i.e. leap seconds are omitted. Time must be derived in SENSORIS based on a **monotonic clock**. It has to be assured that time never jumps, i.e. positions and time are continuous. Therefore automatic adjustment of the clock based on e.g. Network Time Protocol (NTP) or phone network time is discouraged.

2.3. Conventions

SENSORIS uses conventions for its versioning scheme and for the naming of protobuf message, field, and value names.

SENSORIS uses a **sequence-based versioning** scheme that denotes the degree of compatibility. The version is defined by a triplet of integers in the format major.minor.patch, e.g. 1.2.4. All numbers start at zero and are incremented by one. The first public release of SENSORIS has the fixed version 1.0.0. Handling of the versioning scheme depends on the message encoding as defined in Section 2.2 and its abilities concerning backward compatibility. The patch number is incremented for changes in documentation of the message schema only. The minor number is incremented for backward compatible changes, i.e. for extension of the message schema with new message types and fields. The major number is incremented for non-backward compatible changes or if it is intended to indicate an important extension of the message schema.

Deprecation of fields is handled in the schema as defined by the protobuf language, i.e. fields are marked with the modifier [deprecated]. Protobuf message types are marked as deprecated as a SENSORIS convention in their comment. In addition to marking fields or message types as deprecated also the version since the deprecation was introduced and the version when a deprecated field or message type will be removed shall be documented. If a deprecated field or message type is replaced by another field or message type then this shall also be documented.

The **naming convention** used in SENSORIS defines how protobuf message, field, and value names are named in the protobuf schema. It follows the Google Protocol Buffers style guide.[7] Message and enum type names follow the upper camel case naming scheme, i.e. they start with an initial upper case letter and each word or abbreviation in a compound name begins with a capital letter, e.g. VehiclePosition. Field names follow the lower case underscore separator naming scheme, i.e. they consist only of lower case letters and words in a compound name are separated by an underscore, e.g. ignition_on. Enum value names follow the capitals with underscores naming scheme, i.e. they consist only of upper case letters and words in a compound name are separated by an underscore, e.g. TURN_LEFT.

The documentation of the SENSORIS schema is part of the protobuf schema itself, i.e. schema definition and documentation are located together. Documentation is written as protobuf comments. The comments in the protobuf schema are taken over automatically to the auto-generated data classes by the protobuf compiler.

2.4. Data Message Content

The architectural parts of the SENSORIS data message are addressed in this section.

Identifiers that relate to privacy aspects are described in Section 2.4.1. Subsequently, identifiers that are used for cross-referencing of events are detailed in Section 2.4.2. Identifiers that are used for referencing events of a data message to corresponding job requests are described in Section 2.4.3. Attribute representation and meta-attributes are listed in Section 2.4.4. Spatial reference systems used in data messages are described in Section 2.4.5.

2.4.1. Identifiers

Several identifiers are used in a SENSORIS message which affect privacy. They allow for identification of a submitter, session, message, vehicle fleet, vehicle, and driver. All identifiers are optional and are a powerful and fine-grained control instrument for ensuring privacy aspects in SENSORIS.

The **submitter** is optional and defines the origin of messages. A submitter consists of a primary and secondary id, type, software version, and hardware version which are all of type string.

The **message identifier** is optional and defines the order of messages. Message identifiers are of type integer and begin with value 1 and are incremented by 1 on each generation of a message. If information sent from one communication partner to another is split into messages, then each of the messages shall be self-contained, i.e. if one of the messages is lost then the others still shall provide meaningful information.

The **session identifier** is optional and is used to mark messages of the same vehicle that belong together. A new session can be created e.g. on engine start, for each navigation route, after a time threshold, or after inactivity for a time threshold. Session identifiers are of type string and shall be globally unique for each submitter. If the session identifier is set, then all messages of a session can be aligned in order of their

generation. This rule also allows for identification of missing messages, e.g. if messages with identifiers 1, 2, and 4 are communicated to the cloud, then the cloud can derive that message with identifier 3 is missing. It is also possible to report the end of a session explicitly. This enables the cloud to perform processes that require the complete session data in a timely manner.

Examples for the identifiers are shown in Figure 6. If only the message id is set as shown in the first example, then each message starts with an identifier of value 1. For the cloud each received message is from a new vehicle, illustrated as a different vehicle colour in Figure 6 (grey instead of black). It is therefore not possible to track an individual vehicle in the cloud only based on message identifiers. The privacy level obtained with message identifiers is anonymization.

The combination of message and session identifier is shown in the second example in Figure 6. Messages with the same session identifier can be aligned in order and allow for processing messages from the same vehicle that belong together. If a new session is started by a vehicle then for the cloud the messages from the new session are from a new vehicle. It is therefore not possible to track an individual vehicle in the cloud only based on message and session identifiers beyond the scope of a session. **The privacy level obtained with session identifiers is at best anonymization and at worst pseudonymization.**

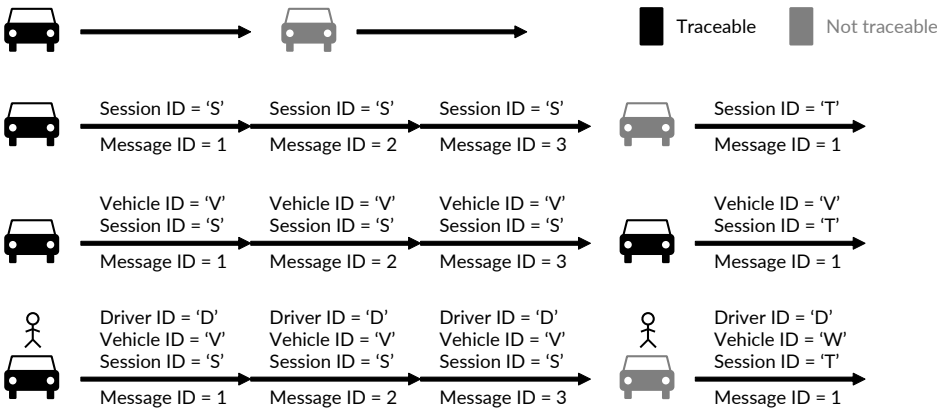


Figure 6. Identifiers and impact on privacy

The **vehicle fleet identifier** is optional and is used to mark messages of vehicles from the same vehicle fleet over the lifetime of the vehicle fleet. Vehicle fleet identifiers are of type string and shall be globally unique for each submitter.

The **vehicle identifier** is optional and is used to mark messages of the same vehicle over its lifetime. Vehicle identifiers are of type string and shall be globally unique for each vehicle fleet. If the vehicle identifier is set, then all messages of a vehicle can be aligned in order of their generation for the complete lifetime of the vehicle. If a vehicle changes its owner, then it shall be considered to either change the vehicle identifier or to reset all data from the vehicle.

The combination of message, session, and vehicle identifiers is shown in the third example in Figure 6. Messages with the same vehicle identifier can be aligned in order and allow for processing messages from the same vehicle across sessions and for its complete lifetime. **The privacy level obtained with vehicle identifiers is pseudonymization.**

The **driver identifier** is optional and is used to mark messages from the same driver over its lifetime. Driver identifiers are of type string and shall be globally unique for each submitter. If the driver identifier is set, then all messages of a driver can be aligned in order of their generation for the complete lifetime of the driver and

for all vehicles the driver has used.

The combination of message, session, vehicle, and driver identifiers is shown in the last example in Figure 6. Messages with the same driver identifier can be aligned in order and allow for processing messages from the same driver across all sessions of all used vehicles for its complete lifetime.

The scope on privacy is handled in Chapter 3.

2.4.2. Identifiers and Referencing

The second set of identifiers is used for cross-referencing events within a message. The order of events within a message is not defined, so forward and backward references are possible. Events are message types of SENSORIS data messages and contain vehicle sensor data.

The **event identifier** uniquely identifies an event within a message and is only required if a reference to the event is needed. Event identifiers are of type integer and begin with value 1 and are incremented by 1.

The **event relation** protobuf message type enables binary relations between events within a single data message.

The **event group** protobuf message type enables smart grouping of events based on the same relative spatial reference system.

The **event source** protobuf message type enables to define the source of a value or an event, via its event identifier.

Some event protobuf message types contain an **object identifier** which enables referencing between individual events over time. For example, as part of the object detection category the same movable object can be referenced over time by its unique object identifier.

2.4.3. Identifiers and Job Requests

The third set of identifiers is used for referencing the events of a data message to the corresponding job request messages (see Section 2.5).

The set of **job request identifiers** of a data message is used to link all events of a data message to the job requests that led to the observation of the events. If a separation of events observed for different jobs is required, then several data messages with events for one job request each can be used.

2.4.4. Attribute Representation and Meta-Attributes

Some attribute types and meta-attributes require modelling of protobuf message types beyond the protobuf scalar value types (see Section 2.2).

The **histogram** protobuf message types enable modelling of histograms with arbitrary sized bins and absolute or relative frequencies. For each of the protobuf scalar value types int64, bool, and string an own protobuf message type is defined. Bin endpoints are modelled with minimal number of fields, i.e. each bin defines only its lower endpoint inclusive value. The upper endpoint inclusive value is defined only for the histogram leading to a collection of bins with value ranges [bin lower endpoint inclusive, next bin lower endpoint

exclusive) with the last bin range [last bin lower endpoint inclusive, upper endpoint inclusive].

Every **enum** field which represents a classification with potential uncertainty is wrapped into a message with a confidence value. The confidence is an integer value in percent with range [0, 100].

All **sensor observations, confidence values, and accuracy values are only relatively comparable to values of the same vehicle**, and then only if all parameters of the complete system remain constant. The derivation of sensor observations, confidence values, and accuracy values in a vehicle, in a vehicle cloud, or in a service cloud may differ e.g. based on sensor hardware, sensor software, environment, or cloud software. SENSORIS explicitly does not cover any alignment of their derivation among different parties.

Relations between events and event source message denote if the event is based on **single sensor observations or sensor fusion**.

2.4.5. Spatial Reference Systems

SENSORIS supports absolute and relative spatial reference systems. **Absolute spatial reference systems** supported by SENSORIS are e.g. World Geodetic System 1984 (WGS84) as described in Section 2.1.

Relative spatial reference systems have a defined origin given in a global spatial reference system. The origin defines the 3D rotation and 3D translation of the relative spatial reference system to the global spatial reference system. The relative position of events to their origin is given in metric distances in x, y, and z axis. Several relative spatial reference systems may overlap and within one data message several relative spatial reference systems may be used.

SENSORIS supports two types of relative spatial reference systems. The first type is the **SENSORIS vehicle coordinate system** as defined in Section 2.1. The relative position of events to their origin can be given by different reference types, which are shown exemplarily in Figure 7. The figure shows a 2D view of the relative spatial coordinate systems. The origin of the SENSORIS vehicle coordinate system is the position P of the vehicle at a timestamp t. In the example, the relative position of a road sign event S at timestamp t to its origin is then given in metric distances on the x and y axis. The different reference types that are possible are described in Table 3.

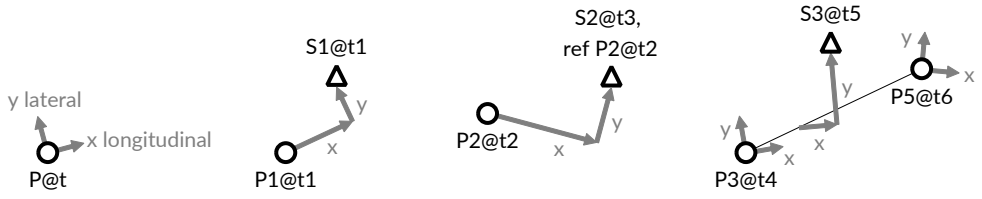


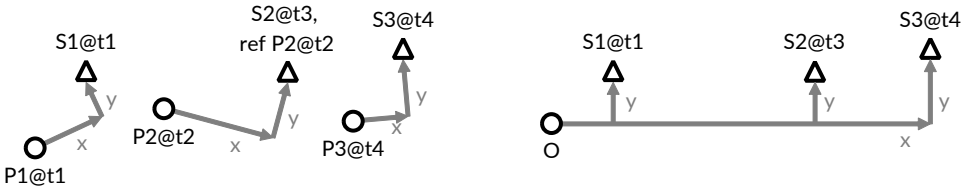
Figure 7. Example for SENSORIS vehicle coordinate system as relative spatial reference system

Reference type	Description
Equal timestamp	Position P1 and sign S1 share the same timestamp t1, therefore the metric distances given for the sign S1 are related to the position P1 as their origin, reference is implicit
Event reference	Position P2 and sign S2 have different timestamps, therefore the reference of the sign to the position has to be explicitly set with an event reference as defined in Section 2.4.2

Reference type	Description
Interpolation	Position P3, sign S3, and position P5 have different timestamps and no event reference is set, therefore the origin of the relative spatial reference system for sign S3 has to be interpolated between positions P3 and P5, interpolation possibly degrades position accuracy of sign S3

Table 3. Reference types for SENSORIS vehicle coordinate system as relative spatial reference system

The second type of relative spatial reference systems supported by SENSORIS is **arbitrary relative spatial reference systems**. Arbitrary refers to the arbitrary position of the origin of the reference systems, which can be different from vehicle positions. The relative position of events to their origin is shown exemplarily in Figure 8. The relative position of events to their origin can be given by different reference types, which are described in Table 4.



Vehicle coord. systems

Arbitrary relative spatial reference system, origin O, either event reference or group

Figure 8. Example for arbitrary relative spatial reference system

Reference type	Description
Event reference	See reference type of SENSORIS vehicle coordinate system in Table 3
Event group	Signs S1, S2, and S3 are explicitly put into the same event group as defined in Section 2.4.2, only one origin per event group is allowed

Table 4. Reference types for arbitrary relative spatial reference system

2.5. Job Request Message Content

The architectural parts of the job request message are addressed in this section.

Job request identifiers and priorities are described in Section 2.5.1. Metadata is described in Section 2.5.2. Section 2.5.3 describes the capability requirements. Overall restrictions are shortly detailed in Section 2.5.4; whereas the validity restrictions are described in Section 2.5.5. The collection triggers are described in Section 2.5.6. Finally the actions that start when the collection triggers are met are described in Section 2.5.7.

2.5.1. Identifiers and Priorities

The **submitter** is optional and defines the origin of messages. A submitter consists of a primary and secondary id, type, software version, and hardware version.

The job request **identifier** is required for all jobs and allows for linking the events of a data message to the corresponding job requests (see also Section 2.4.3).

The job request **priority** is optional and defines the relative importance of job requests. The priority is a numeric value in the range [1, 256], with 1 being the highest and 256 being the lowest priority. Vehicles of a vehicle fleet or a vehicle cloud can use the priority as a hint of which job requests can be cancelled first. For

example, when a vehicle is running out of resources, the vehicle may deactivate the jobs with lowest priority. Job priorities are part of the job metadata described in Section 2.5.2.

The **consolidation of job requests** for a vehicle fleet is performed by the corresponding vehicle cloud. The vehicle cloud consolidates both internal jobs and jobs from connected service clouds in terms of content and priority. It may change and align priorities and combine job requests if they are compatible. A vehicle cloud may also refuse jobs, e.g. due to sensor and resource availability in the vehicles of the vehicle fleet or based on the contract with a service cloud.

An example for the consolidation of job requests is shown in Figure 9. The vehicle cloud consolidates the job request from service cloud A asking for positions every five seconds and the job request from service cloud B asking for positions every ten seconds. The two job requests are consolidated into one job request asking for positions every five seconds. The events of the data messages sent from the vehicle fleet to the vehicle cloud are then routed to the service clouds A and B, whereby service cloud A receives all position events and service cloud B receives only every second position event.

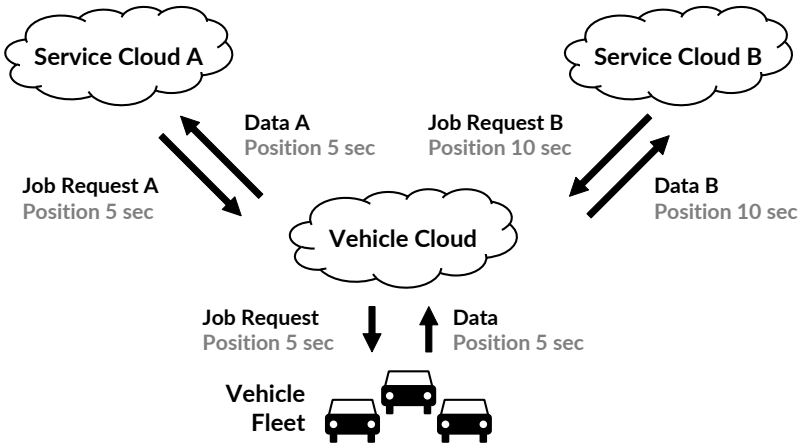


Figure 9. Example for consolidation of job requests by vehicle cloud

A vehicle cloud can also **split job requests** in different sub jobs for a vehicle fleet. The vehicle cloud may receive a job from a connected service cloud. The vehicle cloud may decide to split before sending them to the vehicles. The vehicle cloud may use any splitting methods as needed (for example, by area, time, or others).

An example for the split of job requests is shown in Figure 10. The vehicle cloud split the job request from the service cloud asking for vehicle data for certain city. The vehicle cloud splits this job in different sub jobs, which correspond to a certain area of the city. Each sub job identifier is the result the concatenation of the original job identifier with the corresponding sub job identifier. The events of the data messages sent from the vehicle fleet to the vehicle cloud are then routed to the service cloud with the original job identifier.

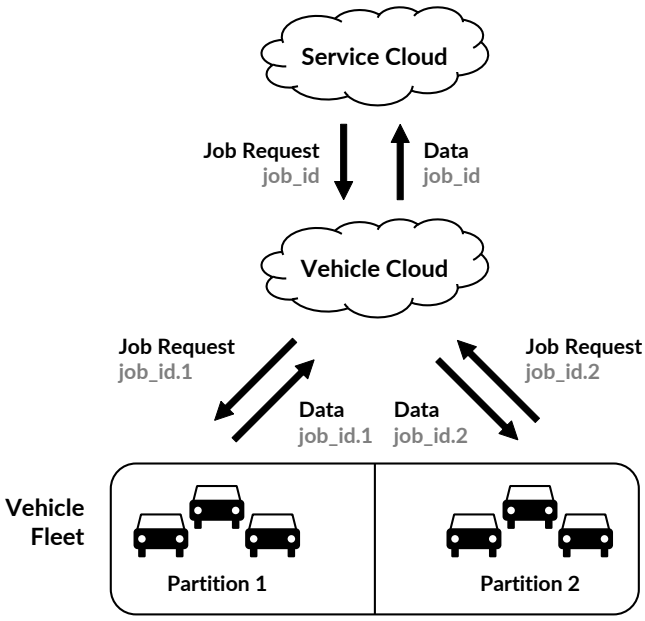


Figure 10. Example for split of job requests by vehicle cloud

2.5.2. Metadata

The job **metadata** provides additional information about the requested data collection, such as job priority (see also Section 2.5.1) or latency requirements for data submission. Job metadata can also provide information if the data collection should take place if vehicles are in accessory mode off (for example, when the vehicle is parked).

2.5.3. Capability Requirements

The job **capability requirements** provide data collectors the basic requirements they need to have in order to interpret and fulfil a job request. These requirements refer to the data messages and extension versions that the submitter of a job request is expecting to receive, and the job request version needed to interpret them.

2.5.4. Overall Restrictions

The job request **overall restrictions** contains the collection restrictions for the entire collection request as a whole, from the service cloud point of view. These are: time restrictions, spatial restrictions and total collection extents. For example, using the total collection extents, the service cloud may restrict the overall number of data messages that is expecting to receive for a job request. The vehicle cloud shall ensure that data messages transmitted to the service cloud do not exceed the specified restriction.

2.5.5. Validity Restrictions

The job **validity restrictions** define conditions under which a job request message is valid, meaning that data collection shall be possible. If the validity restrictions are not met, data collection shall not happen. These restrictions are: time restrictions, spatial restrictions, and map attribute restrictions.

- **Time restrictions** define the temporal conditions for the validity of a job request. These are weekday, date range, and time of the day range.
- **Spatial restrictions** define spatial conditions for the validity of a job request. These can be circle, rectangle, polygon, and directed corridor.

- **Map attribute restrictions** define restrictions over a particular map attribute. These restrictions are map provider independent and they are based on URN (Uniform Resource Name). For example, a service cloud may request data collection in certain road classes. In addition, this type of restriction may be optional, meaning that if the vehicles do not have the required map version to interpret this field, they can ignore the restriction.

2.5.6. Collection Trigger

The **collection trigger** defines in what moment the data collection (action, as described in Section 2.5.7) shall start. Collection triggers are based on logical expressions. They also define the maximum extents for a collection action, meaning that collection action shall stop when these extents are reached.

2.5.7. Actions

The **actions** for job requests define what action shall be performed when the collection trigger conditions are true. In SENSORIS v1.1.0 actions are data collection actions, which define factors as what data shall be collected, how much, how often, etc.

2.6. Job Status Message Content

The architectural parts of the SENSORIS job status message are addressed in this section.

The job status message contains the status of job requests. The status can be monitored and analysed by the requesting vehicle or service cloud. The job status message contains information related to the **termination** of a job request message.

Each job has a **job state**, which is described in Section 2.6.1. In addition a job status message has an optional textual description which can be used to give more details on the job status, e.g. why a job has been terminated.

2.6.1. Job States

The possible states of a job are shown in the Unified Modelling Language (UML) state diagram in Figure 11. Each newly created job is validated first. Validation criteria may cover, amongst others, aspects of privacy, security, and capability requirements. Detailing validation criteria is out of scope of SENSORIS. If validation of the job fails or an exception occurs during validation, then the job state is **Terminated**. If validation of the job is successful, then the job state is **Inactive**. If the job validity restrictions and collection trigger conditions are met, the job state changes from Inactive to **Active**. If the job validity restrictions and collection trigger conditions are no longer met, e.g. by leaving the job spatial restriction, then the job state changes back from Active to **Inactive**. If the job is complete, e.g. by maxing out the temporal interval of the job or by reaching the defined total number of extents, then the job state changes from either Inactive or Active to Terminated. If an exception occurs while the job state is either Inactive or Active, then the job state changes also to Terminated.

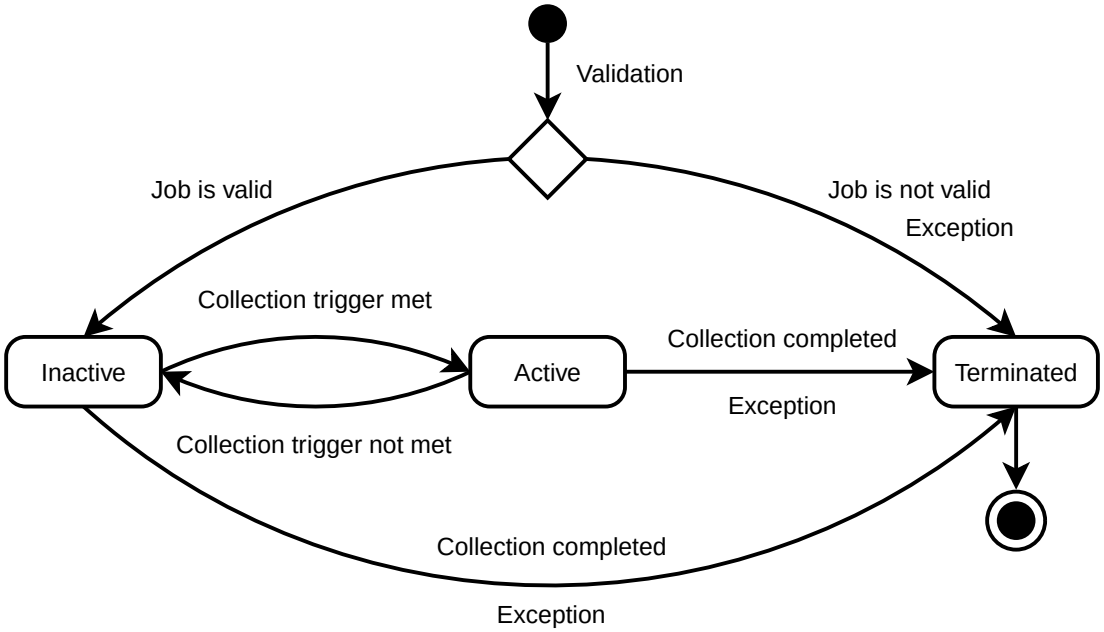


Figure 11. Job states

[1] See http://www.bipm.org/en/publications/si-brochure/, 8th edition from 2014
[2] ISO 8855:2011 Road vehicles — Vehicle dynamics holding and road- ability — Vocabulary
[3] See e.g. https://en.wikipedia.org/wiki/Comparison_of_data_serialization_formats for an extensive list
[4] See https://developers.google.com/protocol-buffers/
[5] See https://github.com/google/protobuf/blob/master/src/google/protobuf/any.proto
[6] See https://github.com/google/protobuf/blob/master/src/google/protobuf/wrappers.proto
[7] See https://developers.google.com/protocol-buffers/docs/style

3. Privacy Handling

SENSORIS is a standard for the exchange of in-vehicle sensor data. This may also include private, personal, pseudonymized, or data that can be derived to personal information. Further, these are named "privacy data". It is the understanding of SENSORIS that privacy data may fall under a governmental protection such as the GDPR. Furthermore, it is the understanding of SENSORIS, that the transportation of any data (including personal data) shall be executed in line with the local regulations. By way of example, these can include the usage of:

- data encryption during transportation
- specific handling at the sending or receiving components of the data
- data owner consent and information (opt-in / opt-out)

The following paragraph contains a list of possible attributes that may require a classification into privacy data, however, it is not complete.

- Anytime
  - a single position point if the single position can identify a person (e.g. on private property). A position point information can be constructed of:
    - a geographic position (longitude, latitude),
    - a relative position (x,y,z) from another known reference point, or
    - a map referenced position e.g. link_id, link_offset
  - the provision of the information on a specific person or a specific vehicle together with additional sensor data. Identification could be:
    - persistent vehicle ID (an identification of a vehicle that does never change over time)
    - persistent driver ID
  - a set of vehicle capabilities including hardware or software sensors, that allow to identify a certain vehicle in a certain region together with additional sensor data. This could include:
    - information about the capability of a specific source (e.g. front RADAR detector)
    - Information about the installed hardware (e.g. sensor supplier)
    - Information about hardware or software version of installed hardware.
- Within one Message
  - a path of multiple positions including the starting and ending point of a given drive. Each position point information can be constructed of:
    - a geographic position (longitude, latitude),
    - a relative position (x,y,z) from another known reference point, or
    - a map referenced position e.g. link_id, link_offset
- Within multiple Messages
  - a temporal id for a person, vehicle, or session where multiple messages could be merged together where individual messages do not meet the definition of privacy data but the identification of multiple messages (from the same vehicle) do.

SENSORIS does not specify

- locations to be handled as private properties,
- the maximum number of positions in a path allowed for non-privacy data or the

- local regions where specific regulations apply.

It is the understanding of SENSORIS that the handling of privacy data is to be executed based on local regulations on a bilateral base between sending and receiving party.

Generally, the SENSORIS specification does not foresee the usage of personal information, such as name, address, etc. of the driver or owner.

Glossary

Term	Description
GNSS	Global Navigation Satellite System
ISO	International Organization for Standardization
OEM	Original Equipment Manufacturer
Protobuf	Google Protocol Buffers
SENSORIS	Sensor Interface Specification
SI	International System of Units
UML	Unified Modeling Language
UTC	Coordinated Universal Time
WG	Working Group
WGS84	World Geodetic System 1984

Table 5. Glossary