GENIVI Alliance

GENIVI Document CS00026

EnhancedPositionService

Component Specification

Accepted Version 4.0.0-alpha

16-Dec-2015

Sponsored by:
GENIVI Alliance

Abstract:
This document provides the Component Specification for the EnhancedPositionService

Keywords:
GENIVI, EnhancedPositionService, GPS, GNSS, Sensors, Dead-Reckoning.

GENIVI Alliance
2400 Camino Ramon, Suite 375
San Ramon, CA 94583, USA

Revision History

The following table shows the revision history for this document.

Document revision history

Date	Version	Author	Description
10-Dec-2014	3.0.0-alpha	Marco Residori, XS Embedded (now part of Mentor Graphics)	Updated API documentation and sequence diagrams. This is the first version of this document that uses the new GENIVI component specification template. Improvements after EG-LBS review
19-Jan-2015	3.0.0-alpha	Helmut Schmidt Continental Automotive GmbH	Update text according remaining review comments
21.Jan-2015	3.0.0	Marco Residori, XS Embedded (now part of Mentor Graphics)	Changed status to "Accepted"
16-Jan-2015	4.0.0-alpha	Marco Residori, Mentor Graphics	Updated API documentation in preparation to Release 4.0.0

Table of Contents

1 Introduction

1.1 System Overview

The GENIVI Software Platform is a platform consisting of standardized middleware, application layer interfaces and frameworks defined or adopted by the GENIVI Alliance.

1.2 Component Overview

The EnhancedPositionService is a software component of the above mentioned GENIVI Software Platform that offers positioning information to client applications.

To calculate the current vehicle position, data from a GNSS receiver (e.g. GPS data) and available vehicle sensors (e.g. gyroscope and wheel ticks) are taken into account (dead-reckoning). In this way the EnhancedPositionService can calculate the current position even on roads, where the GNSS signal is too weak (e.g. in a tunnel, or in a parking garage) or too inaccurate (e.g. in a city or in a canyon).

1.3 Document Overview

This document describes the architecture and the interface of the GENIVI EnhancedPositionService.

2 References

The following standards and specifications contain provisions, which through reference in this document constitute provisions of this specification. All the standards and specifications listed are normative references. At the time of publication, the editions indicated were valid. All standards and specifications are subject to revision, and parties to agreements based on this specification are encouraged to investigate the possibility of applying the most recent editions of the standards and specifications indicated below.

[1] "GENIVI GNSSService – Component Specification" - http://git.projects.genivi.org/?p=lbs/positioning.git;a=tree;f=gnss-service/doc

[2] "GENIVI SensorsService – Component Specification" – http://git.projects.genivi.org/?p=lbs/positioning.git;a=tree;f=sensors-service/doc

[3] GENIVI UML Model - https://svn.genivi.org/uml-model/genivi/trunk

3 Glossary

Acronym	Term	Definition
GNSS	Global Navigation Satellite System	GNSS is a space-based satellite navigation system that provides location and time information.
GPS	Global Positioning System	GPS is a space-based GNSS maintained by the United States government.
GLONASS	Globalnaya navigatsionnaya sputnikovaya sistema	GLONASS is a space-based GNSS operated by the Russian Aerospace Defence Forces.
BDS	BeiDou Navigation Satellite System	BDS is a Chinese GNSS, also known as COMPASS.
	Galileo	Galileo is a GNSS currently being built by the European Union (EU) and European Space Agency (ESA).
	Vehicle Sensors	Vehicle sensors are sensors used for positioning calculation which are located either in the vehicle itself or directly in the unit where the EnhancedPositionService is deployed. Examples are Gyroscopes, Accelerometers, wheel tick or vehicle speed sensors.
DR	Dead Reckoning	In strict sense: A technique that calculates the current position of a vehicle by integrating the relative changes in heading and distance over time since leaving a known starting point. The starting point can be determined e.g. from a GNSS system and the heading and distance changes can be determined from the vehicle sensors. In a more common sense: The fusion of GNSS and vehicle sensor data to calculate improved position and velocity. I.e. even when a GNSS fix is available.

Table 1 – Acronym and Term Definitions

4 Requirements

The requirements related to the EnhancedPositionService are located in the GENIVI UML model (see [3]) in the package *GENIVI Model/LogicalView/SW Platform requirements/Location Based Services/Positioning.*

5 Constraints and Assumptions

This is a handwritten chapter that summarizes the constraints and assumptions done in the project for the component.

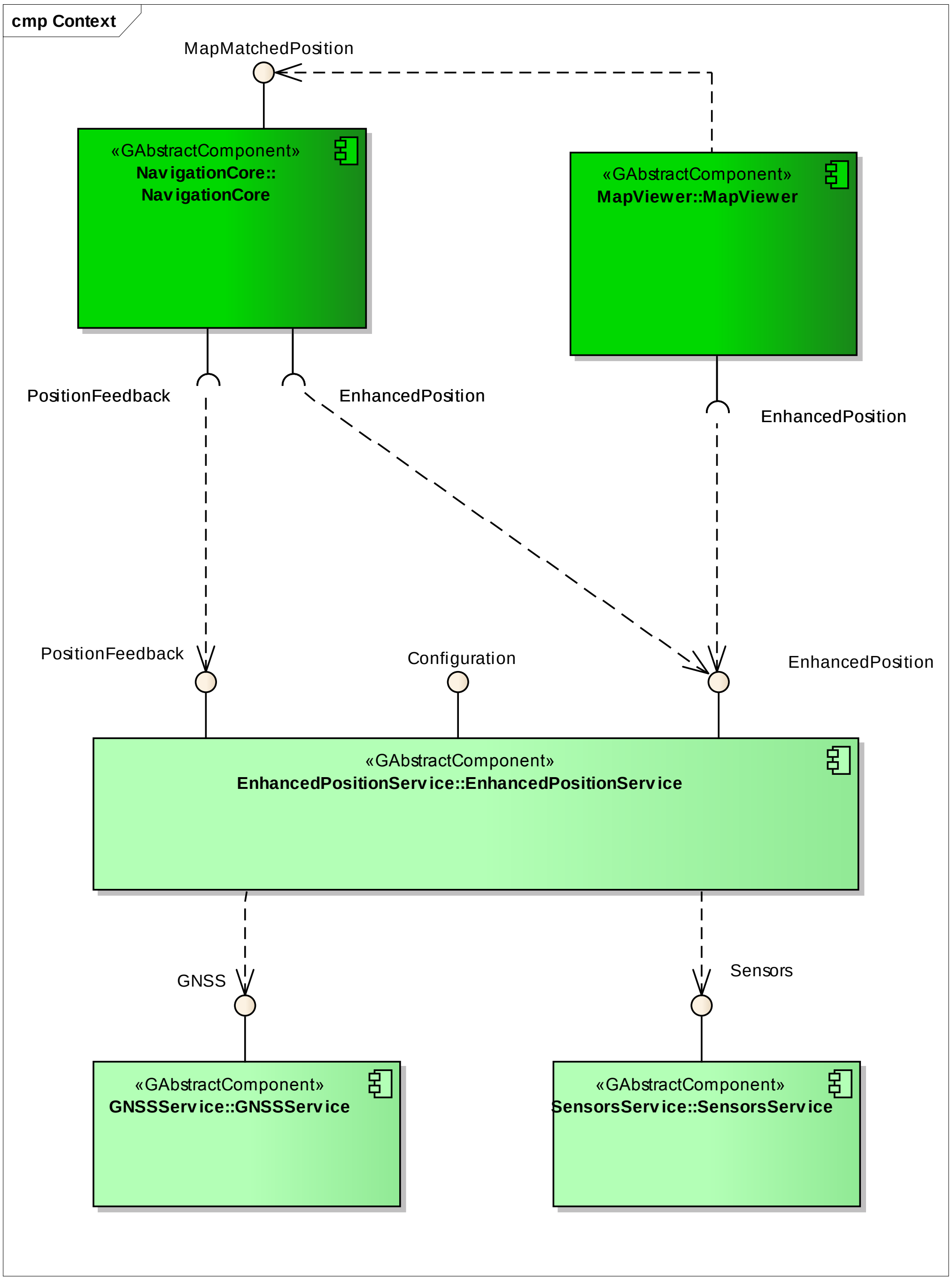
6 Architecture

The information in this chapter is provided only for information purpose; this is not a normative part.

6.1 Architecture Overview

The following component diagram shows how the EnhancedPositionService interacts with other GENIVI components:

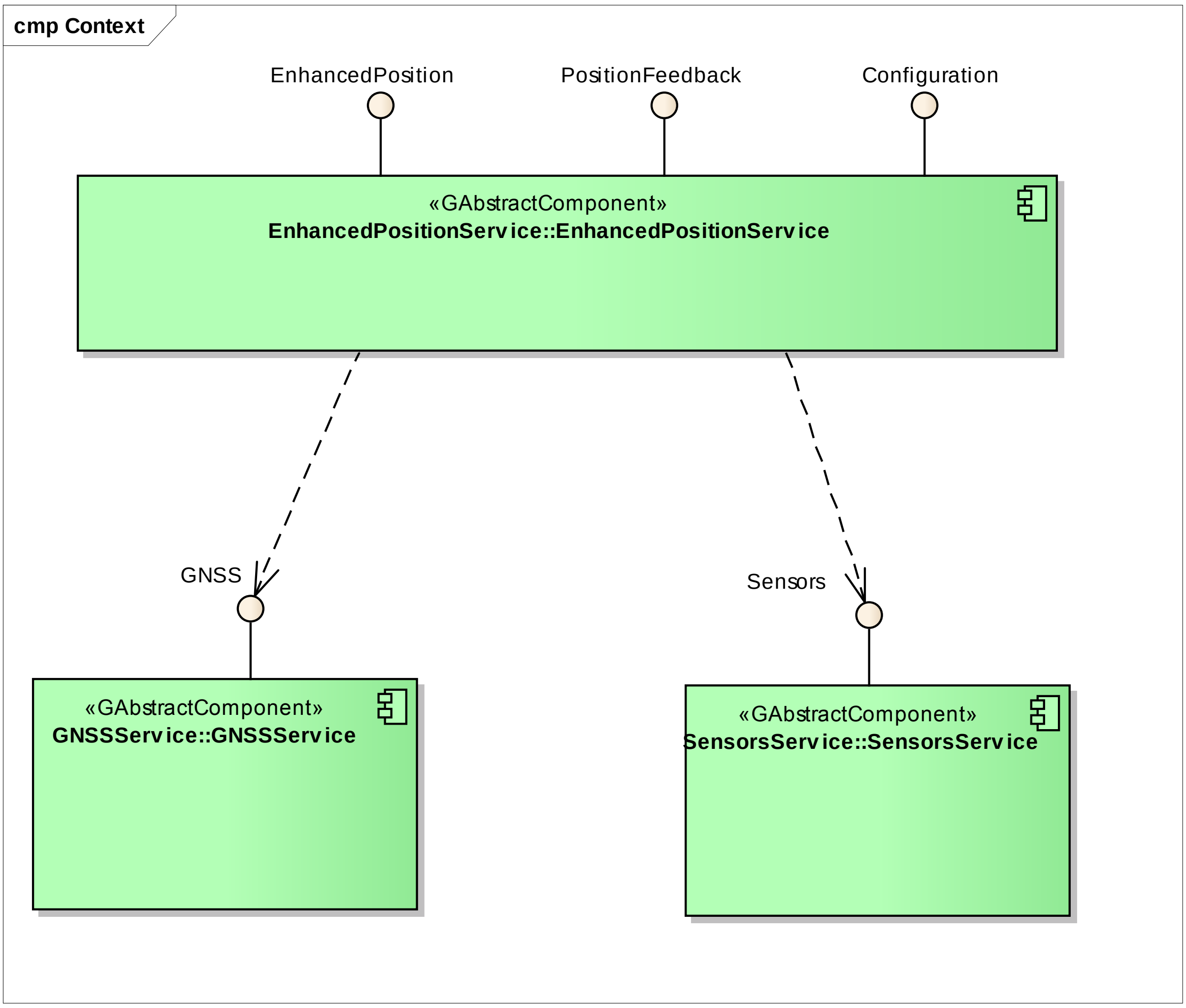
- GNSSService (C library)
- SensorsService (C library)
- NavigationCore (example of client application)
- MapViewer (example of client application)



6.1.1 Component Dependencies

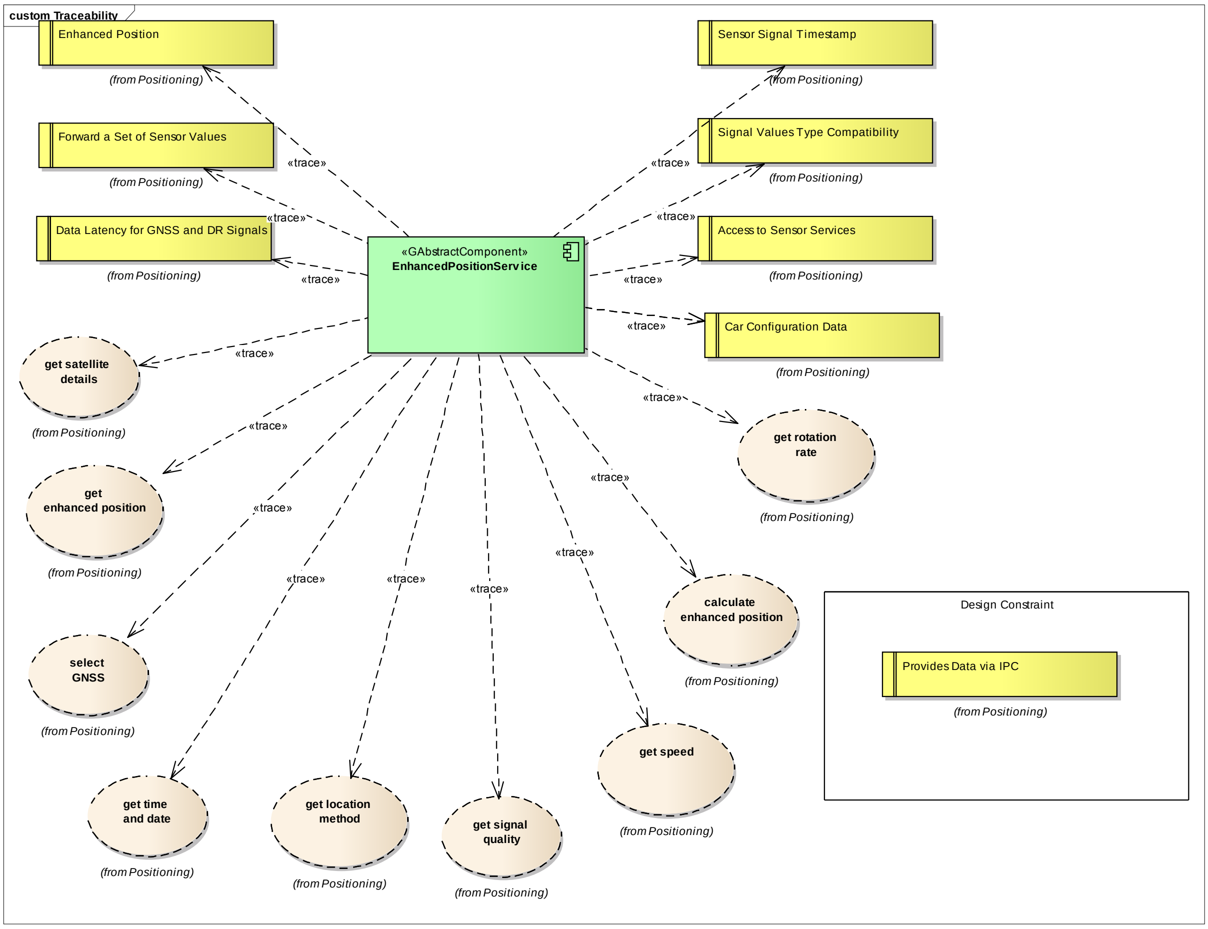
The EnhancedPositionService depends on the following GENIVI components:

- GNSSService (library)
- SensorsService (library)

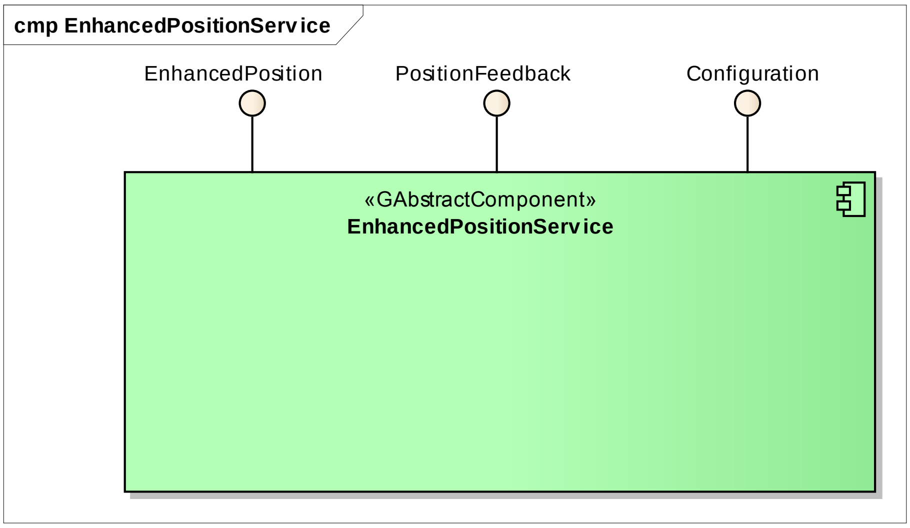


6.1.2 Component Traceability

The following diagrams shows to which requirements and use cases realizations the EnhancedPositionService is traced to:



6.2 EnhancedPositionService



6.2.1 Responsibility and Features

The EnhancedPositionService is a software component that offers positioning information to client applications.

To calculate the current vehicle position, data from a GNSS receiver (e.g. GPS data) and available vehicle sensors (e.g. gyroscope and wheel ticks) are taken into account (dead-reckoning). In this way the EnhancedPositionService can calculate the current position even on roads, where the GNSS signal is too weak (e.g. in a tunnel, or in a parking garage).

The result of the map matching can be provided as feedback to this module by the NavigationCore component. This component is the main client of the GNSSService and of the SensorsService.

The EnhancedPositionService will be typically implemented as a multi-client daemon with a D-Bus interface.

6.2.2 Provided Interfaces

• **EnhancedPosition**: This interface provides a ’filtered’ position that takes into account the value coming from the vehicle sensors (dead-reckoning).

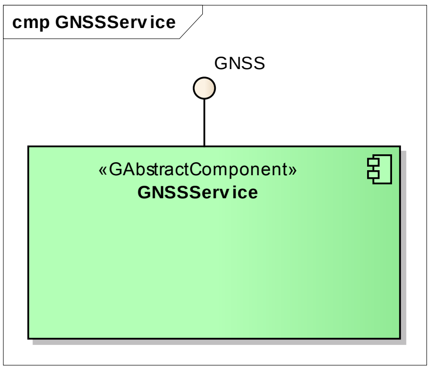
• **PositionFeedback**: This interface offers methods that allows the NavigationCore to provide a position feedback to the EnhancedPositionService. The component that implements the Position-Feedback interface requires the data provided by a ‘map matcher‘ (typically the NavigationCore component). The PositionFeedback is an added improvement which does not negatively affect systems that don´t support maps or have a map-matching feature.

• **Configuration**: This interface allows a client application to manage configuration parameters, like the GNSS type.

6.2.3 Required Interfaces

- **GNSS**: This interface abstracts the access to a GNSS device. Please see [1].
- **Sensors**: This interface abstracts the access to vehicle sensors. Please see [2].

6.3 GNSSService



6.3.1 Responsibility and Features

The GNSSService is a component that retrieves positioning data from a GNSS receiver (e.g. NMEA sentences from a GPS receiver) and presents them to its client applications.

The GNSSService will be typically implemented as a single-client library.

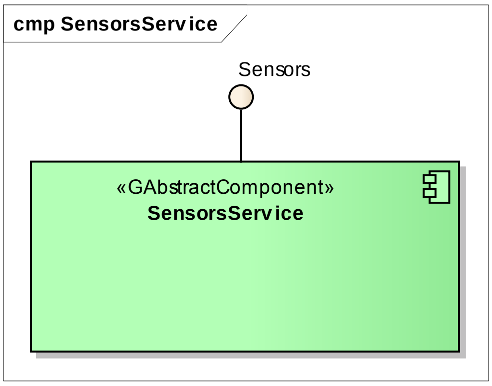
6.3.2 Provided Interfaces

The interfaces provided by this component are described at [1].

6.3.3 Required Interfaces

None.

6.4 SensorsService



6.4.1 Responsibility and Features

The SensorsService is a component that retrieves sensor data from several vehicle sensors (e.g. gyroscope, wheel ticks) and presents them to its client applications.

The SensorsService will be typically implemented as a single-client library.

6.4.2 Provided Interfaces

The interfaces provided by this component are described at [2].

6.4.3 Required Interfaces

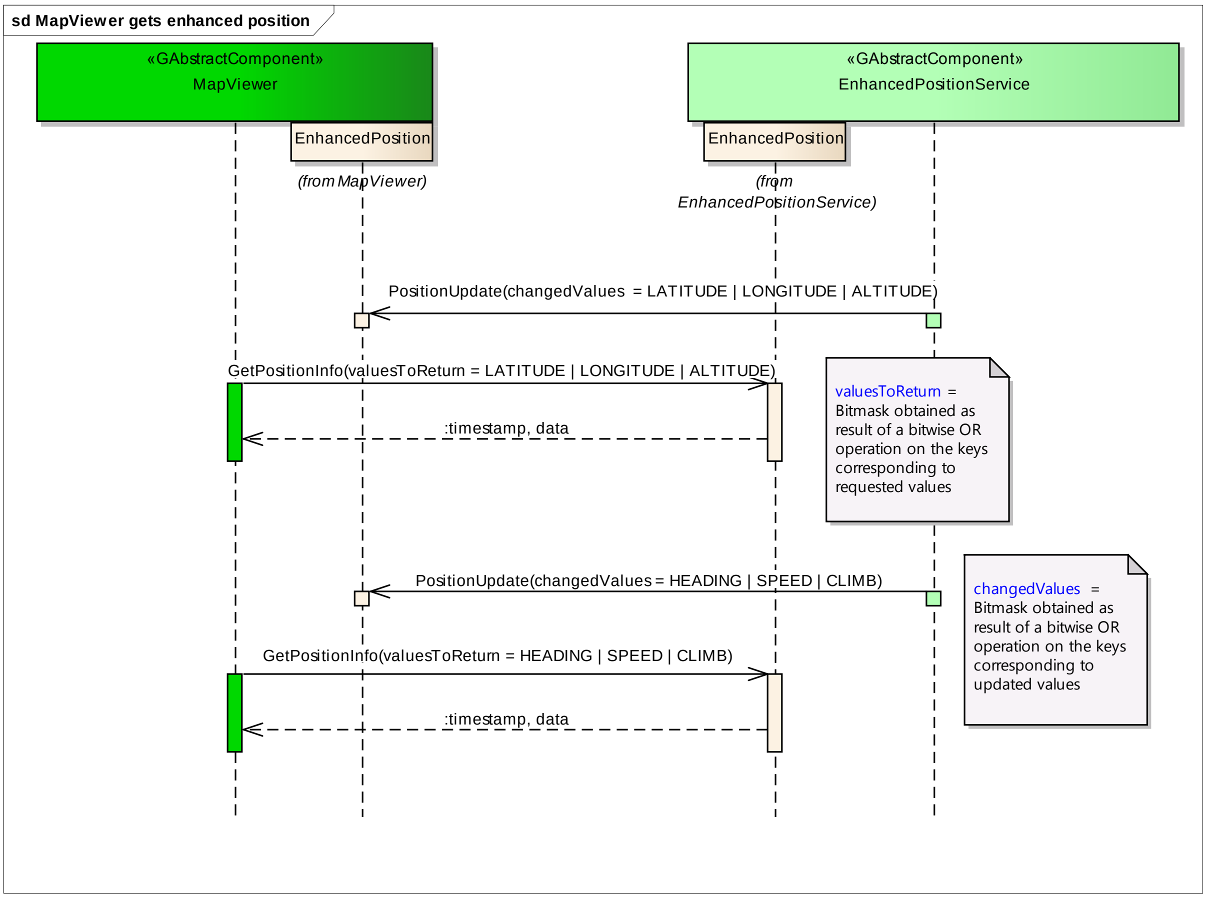
None.

7 Collaboration

7.1 Get Enhanced Position

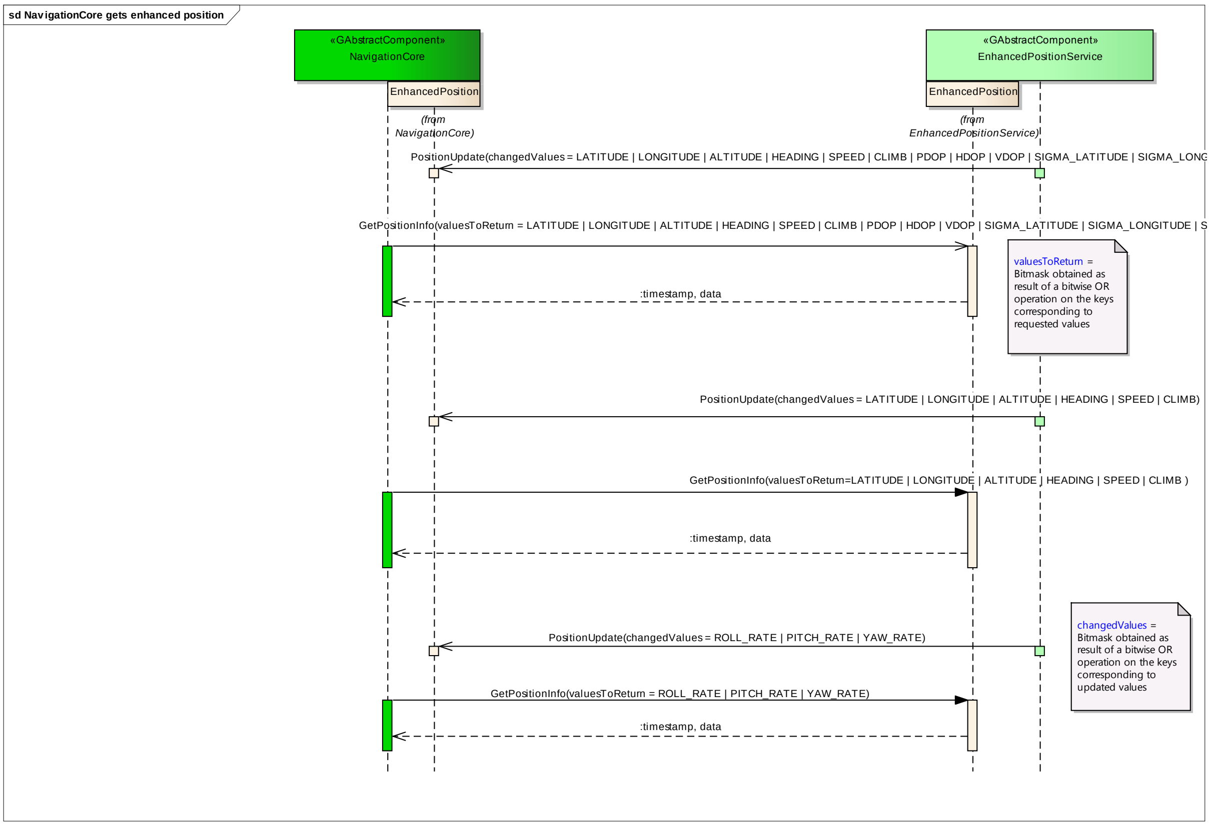
7.1.1 MapViewer retrieves enhanced position

The following sequence diagram describes how a client application can retrieve the vehicle position.



7.1.2 NavigationCore retrieves enhanced position

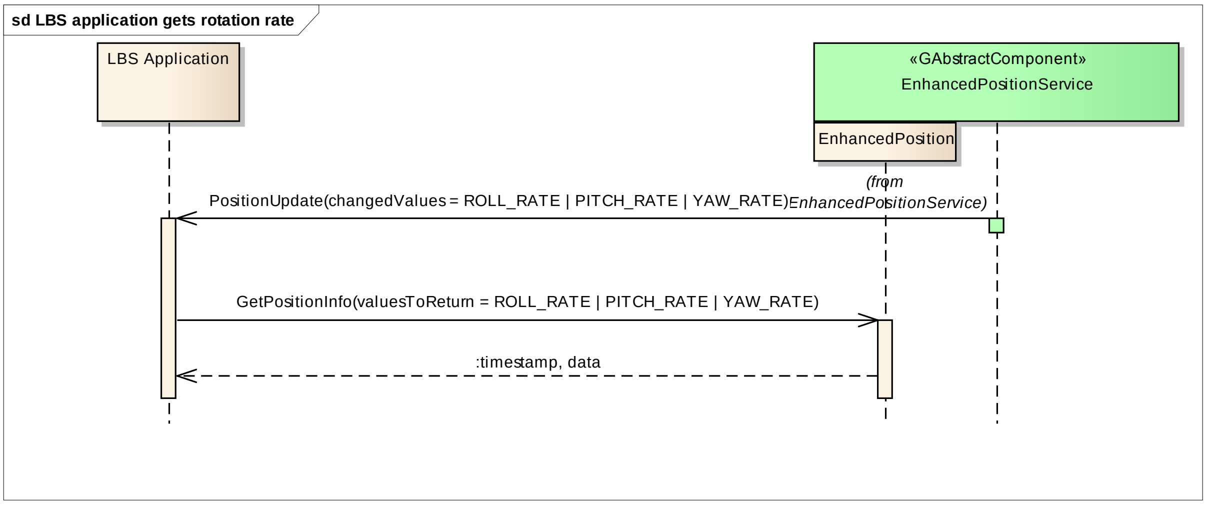
The following sequence diagram describes how a client application can retrieve the vehicle position.



7.2 Get Rotation Rate

7.2.1 LBS Application retrieves rotation rate

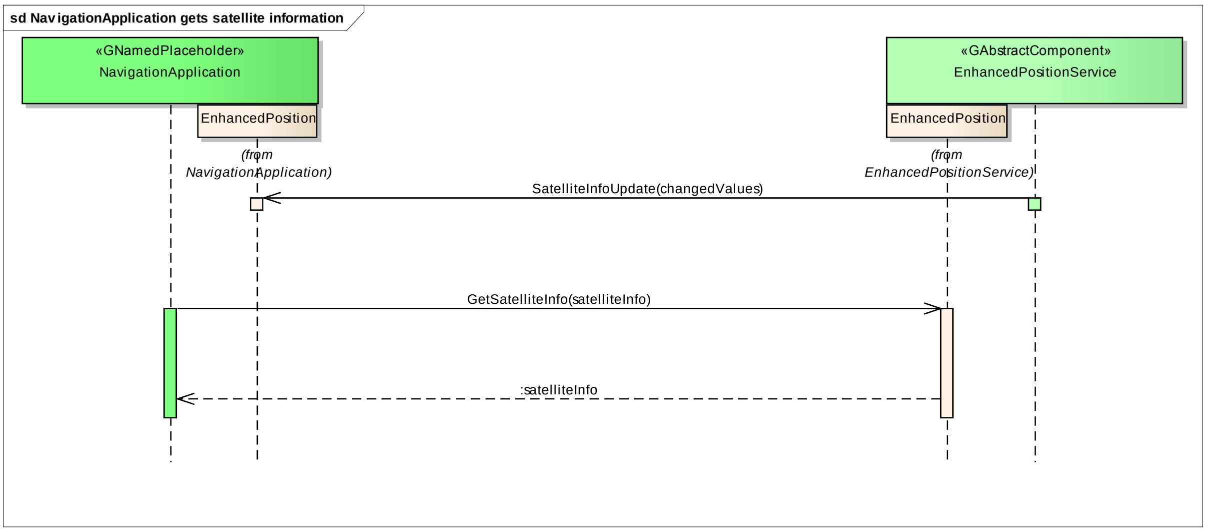
The following sequence diagram describes how a client application can retrieve the vehicle rotation rate.



7.3 Get Satellite Details

7.3.1 Navigation Application retrieves satellite information

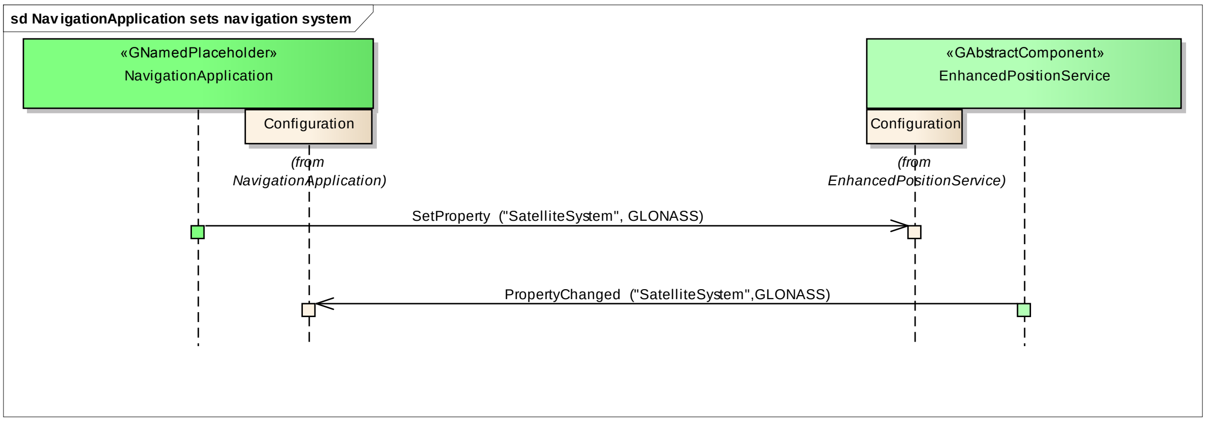
The following sequence diagram describes how a client application can retrieve satellite information.



7.4 Set Navigation System

7.4.1 Navigation Application sets navigation system

The following sequence diagram describes how a client application can set the satellite system.



8 Implementation

8.1 Available Implementation details

A Proof of concept (PoC) of the EnhancedPositionServiceis is available at:
http://git.projects.genivi.org/?p=lbs/positioning.git;a=tree

8.2 Usage examples

Please see: http://git.projects.genivi.org/?p=lbs/positioning.git;a=tree;f=enhanced-position-service/test.

8.3 Test Plan

Please see: http://git.projects.genivi.org/?p=lbs/positioning.git;a=blob;f=enhanced-position-service/doc/testplan.txt

9 Interfaces

The following pages describe the interfaces of the EnhancedPositionService.

9.1 D-Bus

The EnhancedPositionService interfaces are D-Bus interfaces. They are defined using the D-Bus introspection data format, which is nothing but an IDL expressed in XML format.

For more information about the D-Bus data types please refer to the following website:
http://dbus.freedesktop.org/doc/dbus-specification.html#message-protocol-signatures

For more information about the D-Bus introspection data format, please refer to the following website:
http://dbus.freedesktop.org/doc/dbus-specification.html#introspection-format

9.2 Git Repository

The EnhancedPositionService interfaces can be found in the GENIVI Git repository at:
http://git.projects.genivi.org/?p=lbs/positioning.git;a=tree;f=enhanced-position-service/api

9.3 Naming Conventions

Please see http://dbus.freedesktop.org/doc/dbus-specification.html.

Element	***Description***	***Example***
Interface File	*genivi.<component name or domain in lowercase character>.<interface name in lowercase characters>*	*org.genivi.positioning.Configuration*
Methods/Signal/Properties	*Camel case naming convention* *First letter uppercase*	*GetPositionInfo*
Arguments	*Camel case naming convention* *First letter lowercase*	*valuesToReturn*

9.4 Data Types Convention

D-bus types code are used. Please refer to the following webpage for more information:
http://dbus.freedesktop.org/doc/dbus-specification.html

Element	***D-Bus Data Type Code***	***Example***
Enumerators	*q (uint16)*	
Handles	*y (uint8)*	
Maps	*a{qv}*	*Dictionary of tuples (key, value)* *The key is expressed as an enumerator*

9.5 Errors

Error Type	Description	Example	Error Documentation	Note
User Error	Error caused by user actions	The user tries to start route guidance, although guidance is already running	Application specific error string documented in the XML file	Can occur in final product
Hardware Error	Error related to hardware/database related problems	No map data	Application specific error string documented in the XML file	Can occur in final product
Protocol Error	Error caused by wrong sequence of commands	Wrong sequence of commands to enter destination	Standard D-Bus error string	Should not occur in final product
Bus Error	D-Bus communication error	Bus busy	Standard D-Bus error string	Can occur in final product
Programming Error	Programming Error	Invalid parameters	Standard D-Bus error string and debug messages	Should not occur in production code

Only application-specific errors are documented directly in the interfaces (XML files). For all other errors, standard D-Bus strings are used. These kinds of strings are not documented in the interfaces. It is implicitly assumed that every method may return a standard D-Bus error string.

Please see http://dbus.freedesktop.org/doc/api/html/group__DBusProtocol.html.

interface *org.genivi.positioning.Configuration*

version 4.0.0alpha (28-Oct-2015)

Configuration = This interface allows a client application to set and retrieve configuration options
For each configuration option, a property is provided
A key identifying a configuration option is a string containing the name of the property
The possible values for each configuration property including their data type are described as part of the documentation of the property

---

GetVersion = This method returns the API version implemented by the server application.
method `GetVersion`

version = struct(major,minor,micro,date)
major = when the major changes, then backward compatibility with previous releases is not granted
minor = when the minor changes, then backward compatibility with previous releases is granted, but something changed in the implementation of the API (e.g. new methods may have been added)
micro = when the micro changes, then backward compatibility with previous releases is granted (bug fixes or documentation modifications)
date = release date (e.g. 21-06-2011)
out **(qqqs)** `version`

GetProperties = This method returns the current values of all global system properties.
method `GetProperties`

properties = array[property]
property = dictionary[key,value]
The possible values for each configuration property including their data type are described as part of the documentation of the property
out **a{sv}** `properties`

SetProperty = This method changes the value of the specified property
Only properties that are listed as read-write are changeable
On success a PropertyChanged signal will be emitted
method `SetProperty`

name = property name
in **s** `name`

value = property value
in **v** `value`

error *`org.genivi.positioning.Configuration.Error.InvalidProperty`*

PropertyChanged = This signal is emitted when a property changes

signal `PropertyChanged`

> *name = property name*
>
> *in* **s** `name`
>
> *value = property value*
>
> *in* **v** `value`

SatelliteSystem = Bitmask obtained as result of a bitwise OR operation on the keys corresponding to the satellite systems (GPS,GLONASS,GALILEO,COMPASS, ...) to be used

property `SatelliteSystem` *readwrite* **u**

UpdateInterval = update interval in ms

property `UpdateInterval` *readwrite* **i**

GetSupportedProperties = This method returns all supported global system properties
For each property, an array of all possible values is provided

method `GetSupportedProperties`

> *properties = array[property]*
> *property = dictionary[key,value]*
> *key = enum(SatelliteSystem,UpdateInterval, ...)*
> *key = SatelliteSystem, value = value of type 'aq'; 'q' is an enum(GPS,GLONASS,GALILEO,COMPASS, ...)*
> *key = UpdateInterval, value = value of type 'ai'; 'i' is the update interval in ms*
>
> *out* **a{sv}** `properties`

interface *org.genivi.positioning.EnhancedPosition*

version 4.0.0alpha (28-Oct-2015)

EnhancedPosition = This interface offers functionalities to retrieve the enhanced position of the vehicle

---

GetVersion = This method returns the API version implemented by the server application
method `GetVersion`

version = struct(major,minor,micro,date)
major = when the major changes, then backward compatibility with previous releases is not granted
minor = when the minor changes, then backward compatibility with previous releases is granted, but something changed in the implementation of the API (e.g. new methods may have been added)
micro = when the micro changes, then backward compatibility with previous releases is granted (bug fixes or documentation modifications)
date = release date (e.g. 21-06-2011)
out **(qqqs)** `version`

GetPositionInfo = This method returns a given set of positioning data (e.g. Position, Course, Accuracy, Status, ...)
method `GetPositionInfo`

valuesToReturn = Bitmask obtained as result of a bitwise OR operation on the keys corresponding to requested values
Keys: LATITUDE,LONGITUDE,ALTITUDE,
HEADING,SPEED,CLIMB,
ROLL_RATE,PITCH_RATE,YAW_RATE,
PDOP,HDOP,VDOP,
USED_SATELLITES,TRACKED_SATELLITES,VISIBLE_SATELLITES,
SIGMA_HPOSITION,SIGMA_ALTITUDE,
SIGMA_HEADING,SIGMA_SPEED,SIGMA_CLIMB,
GNSS_FIX_STATUS,DR_STATUS,USED_SATELLITESYSTEMS
in **t** `valuesToReturn`

timestamp = Timestamp of the acquisition of the position data [ms]
Note: All timestamps must be based on the same time source.
out **t** `timestamp`

data = dictionary[key,value]
dictionary = array of tuples (key,value)
Invalid data is not be returned to the client application
The vehicle axis system is defined by ISO 8855: In short, the X-axis pointing is forwards, the Y-axis is pointing left, the Z-axis is pointing upwards
key = enum(LATITUDE,LONGITUDE,ALTITUDE,HEADING,SPEED,CLIMB,ROLL_RATE,PITCH_RATE,YAW_RATE,PDOP,HDOP,VDOP,USED_SATELLITES,TRACKED_SATELLITES,VISIBLE_SATELLITES,SIGMA_HPOSITION,SIGMA_ALTITUDE,SIGMA_HEADING,SIGMA_SPEED,SIGMA_CLIMB,GNSS_FIX_STATUS,DR_STATUS,USED_SATELLITESYSTEMS)
key = LATITUDE, value = value of type 'd', that expresses the WGS84 latitude of the current position in degrees. Range [-90:+90]. Example: 48.053250
key = LONGITUDE, value = value of type 'd', that expresses the WGS84 longitude of the current position in degrees. Range [-180:+180]. Example: 8.324500
key = ALTITUDE, value = value of type 'd', that expresses the altitude above the sea level of the current position in meters
key = HEADING, value = value of type 'd', that expresses the course angle in degree. Range [0:360]. 0 = north, 90 = east, 180 = south, 270 = west
key = SPEED, value = value of type 'd', that expresses speed measured in m/s. A negative value indicates that the vehicle is moving backwards
key = CLIMB, value = value of type 'd', that expresses the road gradient in degrees. Range [-180:+180]. A positive value means upwards.
key = ROLL_RATE, value = value of type 'd', rotation rate around the X-axis in degrees/s. Range [-100:+100]
key = PITCH_RATE, value = value of type 'd', rotation rate around the Y-axis in degrees/s. Range [-100:+100]
key = YAW_RATE, value = value of type 'd', rotation rate around the Z-axis in degrees/s. Range [-100:+100]
key = PDOP, value = value of type 'd', that represents the positional (3D) dilution of precision
key = HDOP, value = value of type 'd', that represents the horizontal (2D) dilution of precision
key = VDOP, value = value of type 'd', that represents vertical (altitude) dilution of precision
key = USED_SATELLITES, value = value of type 'y', that represents the number of used satellites
key = TRACKED_SATELLITES, value = value of type 'y', that represents the number of tracked satellites
key = VISIBLE_SATELLITES, value = value of type 'y', that represents the number of visible satellites
key = SIGMA_HPOSITION, value = value of type 'd', that represents the standard error estimate of the horizontal position in m
key = SIGMA_ALTITUDE, value = value of type 'd', that represents the standard error estimate of the altitude in m
key = SIGMA_HEADING, value = value of type 'd', that represents the standard error estimate of the heading in degrees
key = SIGMA_SPEED, value = value of type 'd', that represents the standard error estimate of the speed in m/s
key = SIGMA_CLIMB, value = value of type 'd', that represents the standard error estimate of the climb in degrees
key = GNSS_FIX_STATUS, value = value of type 'q', that represents an enum(NO_FIX(0x00),TIME_FIX(0x01),2D_FIX(0x02),3D_FIX(0x03), ...)
key = DR_STATUS, value = value of type 'b', where TRUE means that a dead-reckoning algorithm has been used to calculate the current position
key = USED_SATELLITESYSTEMS, value = value of type 'u', that represents an Bitmask obtained as result of a bitwise OR operation on the keys corresponding to the satellite systems that are actually used for the position fix
out **a{tv}** `data`

PositionUpdate = This signal is called to notifiy a client application that updated positioning data is available. The update frequency is implementation specific. The maximum allowed frequency is 10Hz
signal `PositionUpdate`

changedValues = Bitmask obtained as result of a bitwise OR operation on the keys corresponding to updated values
LATITUDE,LONGITUDE,ALTITUDE,
HEADING,SPEED,CLIMB,
ROLL_RATE,PITCH_RATE,YAW_RATE,
PDOP,HDOP,VDOP,
USED_SATELLITES,TRACKED_SATELLITES,VISIBLE_SATELLITES,
SIGMA_HPOSITION,SIGMA_ALTITUDE,
SIGMA_HEADING,SIGMA_SPEED,SIGMA_CLIMB,
GNSS_FIX_STATUS,DR_STATUS
in **t** `changedValues`

GetSatelliteInfo = This method returns information about the current satellite constellation
method `GetSatelliteInfo`

timestamp = Timestamp of the acquisition of the satellite detail data [ms]
Note: All timestamps must be based on the same time source.
out **t** `timestamp`

satelliteInfo = array(struct(system,satelliteId,azimuth,elevation,snr,inUse))
system = enum(GPS, GLONASS, GALILEO, BEIDOU, ...)
satelliteId = satellite ID. This ID is unique within one satellite system
azimuth = satellite azimuth in degrees. Value range 0..359
elevation = satellite elevation in degrees. Value range 0..90
snr = SNR (C/No) in dBHz. Range 0 to 99, null when not tracking
inUse = flag indicating if the satellite is used for the fix (inUse=true)
out **a(qqqqqb)** `satelliteInfo`

GetTime = This method returns time and date, typically according UTC time scale
method `GetTime`

timestamp = Timestamp of the acquisition of the date/time [ms]
Note: All timestamps must be based on the same time source.
out **t** `timestamp`

time = dictionary[key,value]
dictionary = array of tuples (key,value)
If you request for a specific value which is invalid, it's not returned in the dictionary.
key = enum(YEAR,MONTH,DAY,HOUR,MINUTE,SECOND,MS,TIME_SCALE,LEAP_SECONDS ...)
key = YEAR, value = value of type 'q', 4 digits number that indicates the year. Example: 2012
key = MONTH, value = value of type 'y', 2 digits number that indicates the month. Example: 03 means March
key = DAY, value = value of type 'y', 2 digits number that indicates the day. Range [0:31]. Example: 07
key = HOUR, value = value of type 'y', 2 digits number that indicates the hour. Range [0:23]. Example: 01
key = MINUTE, value = value of type 'y', 2 digits number that represents the minutes. Range [0:59]. Example: 01
key = SECOND, value = value of type 'y', 2 digits number that represents the seconds. Range [0:59], for leap seconds, also 60 is allowed. Example: 01
key = MS, value = value of type 'q', 3 digits number that represents the milliseconds. Range [0:999]. Example: 007
key = TIME_SCALE, value = = enum(TIME_SCALE_UTC,TIME_SCALE_GPS). Time scale on which the date/time data are based. Example: TIME_SCALE_UTC (0)
key = LEAP_SECONDS, value = value of type 'n', Number of leap seconds, i.e. difference between GPS time and

UTC. Example: 17

out **a{tv}** `time`

interface org.genivi.positioning.PositionFeedback version 3.0.0 (10-Dec-2014)

PositionFeedback = This interface allows the application implementing the map-matching algorithm to provide a position feedback to the EnahncedPositionService

---

GetVersion = This method returns the API version implemented by the server application

method `GetVersion`

version = struct(major,minor,micro,date)
major = when the major changes, then backward compatibility with previous releases is not granted
minor = when the minor changes, then backward compatibility with previous releases is granted, but something changed in the implementation of the API (e.g. new methods may have been added)
micro = when the micro changes, then backward compatibility with previous releases is granted (bug fixes or documentation modifications)
date = release date (e.g. 21-06-2011)

out **(qqqs)** `version`

SetPositionFeedback = This method allows a client application to provide the EnhancedPositionService with a position feedback
Note: This interface is typically used by the application that implements the map-matching algorithm
Such application can hand over to the EnhancedPositionService an array of map-matched positions with different values of reliability

method `SetPositionFeedback`

feedback = array[position]
position = dictionary[key,value]
dictionary = array of tuples (key,value)
key = enum(LATITUDE,LONGITUDE,ALTITUDE,HEADING,SPEED,CLIMB,RELIABILTY_INDEX, ...)
key = LATITUDE, value = value of type 'd', that expresses the WGS84 latitude of the current position in degrees. Range [-90:+90]. Example: 48.053250
key = LONGITUDE, value = value of type 'd', that expresses the WGS84 longitude of the current position in degrees. Range [-180:+180]. Example: 8.324500
key = ALTITUDE, value = value of type 'd', that expresses the altitude above the sea level of the current position in meters
key = HEADING, value = value of type 'd', that expresses the course angle in degree. Range [0:360]. 0 = north, 90 = east, 180 = south, 270 = west
key = SPEED, value = value of type 'd', that expresses speed measured in m/s. A negative value indicates that the vehicle is moving backwards
key = CLIMB, value = value of type 'd', that expresses the road gradient in degrees. Range [-180:+180]. A positive value means upwards.
key = RELIABILTY_INDEX, value = value of type 'y', that indicates the position feedback reliability. It can assume values from 0 to 100. Higher values indicate higher reliability.

in **aa{tv}** `feedback`

timestamp = Original timestamp of the corresponding position data received from the EnhancedPosition API [ms]
Note: All timestamps must be based on the same time source.

in **t** `timestamp`

feedbackType = enum(INVALID,MAP_MATCHED_FEEDBACK,TEST_FEEDBACK, ...)

in **q** `feedbackType`

constants ***EnhancedPositionService*** **version 4.0.0alpha (28-Oct-2015)**

- *This document defines the constants that are used in the EnhancedPositionService APIs*

Constants for "Keys" are always individual bits within a 64 bit unsigned integer and are unique within the EnhancedPositionService
Constants for "Enums" increment consecutively and are only unique within the context of the specific enum

---

- LATITUDE = 0x00000001

- LONGITUDE = 0x00000002

- ALTITUDE = 0x00000004

- HEADING = 0x00000008

- SPEED = 0x00000010

- CLIMB = 0x00000020

- ROLL_RATE = 0x00000040

- PITCH_RATE = 0x00000080

- YAW_RATE = 0x00000100

- PDOP = 0x00000200

- HDOP = 0x00000400

- VDOP = 0x00000800

- USED_SATELLITES = 0x00001000

- TRACKED_SATELLITES = 0x00002000

- VISIBLE_SATELLITES = 0x00004000

- SIGMA_HPOSITION = 0x00008000

- SIGMA_ALTITUDE = 0x00010000

- SIGMA_HEADING = 0x00020000

- SIGMA_SPEED = 0x00040000

- SIGMA_CLIMB = 0x00080000

- GNSS_FIX_STATUS = 0x00100000

- DR_STATUS = 0x00200000

- RELIABILTY_INDEX = 0x00400000

- USED_SATELLITESYSTEMS = 0x00800000
- YEAR = 0x01000000
- MONTH = 0x02000000
- DAY = 0x04000000
- HOUR = 0x08000000
- MINUTE = 0x10000000
- SECOND = 0x20000000
- MS = 0x40000000
- TIME_SCALE = 0x80000000
- LEAP_SECONDS = 0x100000000
- INVALID = 0x00000000
- GPS = 0x00000001
- GLONASS = 0x00000002
- GALILEO = 0x00000004
- BEIDOU = 0x00000008
- COMPASS = 0x00000008
- GPS_L2 = 0x00000010
- GPS_L5 = 0x00000020
- GLONASS_L2 = 0x00000040
- BEIDOU_B2 = 0x00000080
- SBAS_WAAS = 0x00010000
- SBAS_EGNOS = 0x00020000
- SBAS_MSAS = 0x00040000
- SBAS_QZSS_SAIF = 0x00080000
- SBAS_SDCM = 0x00100000
- SBAS_GAGAN = 0x00200000
- MAP_MATCHED_FEEDBACK = 0x00000001
- TEST_FEEDBACK = 0x00000002

- TIME_SCALE_UTC = 0

- TIME_SCALE_GPS = 1