



GENIVI NavigationCore API

Release 3.0.1
Status: Approved

22 May 2014

Accepted for release by:

This document has been accepted for the GENIVI Gemini Release by the Expert Group Location Based Services (EG-LBS)

Abstract:

This document describes the API of the NavigationCore Abstract Component.

Keywords:

NavigationAPIs, NavigationCore.

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

Copyright (C) 2012, BMW Car IT GmbH, Continental Automotive GmbH, PCA Peugeot Citroën, XS Embedded GmbH

This work is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License

To view a copy of this license, visit
<http://creativecommons.org/licenses/by-sa/4.0/>
or send a letter to Creative Commons, 444 Castro Street, Suite 900,
Mountain View, California, 94041, USA.

Table of contents

1	Change History	4
2	Introduction	5
3	Terminology	6
4	Requirements	7
5	Architecture	8
5.1	Interfaces	8
5.2	Interaction with other Components	9
6	API	10
6.1	D-Bus	10
6.2	Git Repository	10
6.3	Naming Conventions	10
6.4	Data Types Convention	11
6.5	Errors	12
6.6	Sessions	13
6.7	Sequence Diagrams	15
6.7.1	navigation application creates route	15
6.7.2	navigation application starts route calculation	16
6.7.3	navigation application gets list of segments	17
6.7.4	navigation application enters destination	18
6.7.5	two clients try to change route preferences of the same route	19
6.7.6	navigation application sets route preferences	20
6.7.7	navigation application sets starting point	21
6.7.8	navigation application sets transportation means	22
6.7.9	navigation application changes waypoints order	23
6.7.10	navigation application enables voice guidance	24
6.7.11	navigation application starts a simulation	25
6.7.12	navigation application starts guidance	26
6.7.13	navigation application stops guidance	27
6.7.14	HMI requests voice instruction	28
6.7.15	navigation application creates location input session	29
6.7.16	navigation application enters location	30
6.7.17	navigation application enters location using speller	31
6.7.18	navigation application enters full address	32
6.8	Interfaces	33

1 Change History

Version	Date	Author	Change
0.1	27 Feb 2012	Marco Residori (XS Embedded)	Document Created.
0.2	19 Mar 2012	Marco Residori (XS Embedded)	Updated sequence diagrams. Updated <i>Interfaces</i> chapter.
0.3	21 Mar 2012	Marco Residori (XS Embedded)	Updated <i>Interfaces</i> chapter.
1.0	22 Mar 2012	Marco Residori (XS Embedded)	System Architecture Team (SAT) approval.
2.0 (beta)	07 Jun 2013	Marco Residori (XS Embedded)	Updated API description. API Version 2.0.
2.0	17 Jun 2013	Marco Residori (XS Embedded)	Updated API description. API fixes: GT-2691, GT-2689, GT-2651. API Version 2.0 (<i>gemini-final</i> tag)
3.0.0	21 Jan 2014	Marco Residori (XS Embedded)	Updated API description. API Version 3.0.0
3.0.1	22 May 2014	Marco Residori (XS Embedded)	Updated copyright notes.

2 Introduction

This document describes the NavigationCore API.

3 Terminology

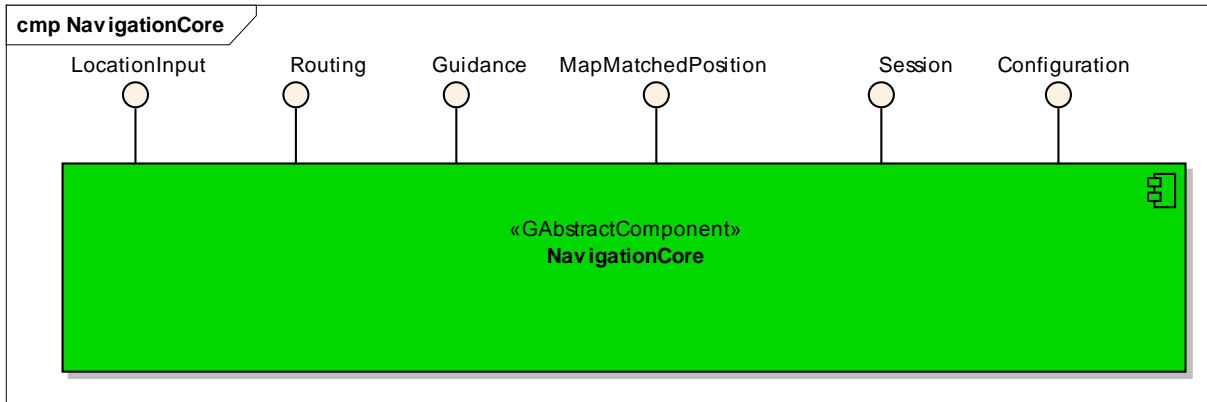
<i>Term</i>	<i>Description</i>
Link-ID	Identifier of a route segment in a database

4 Requirements

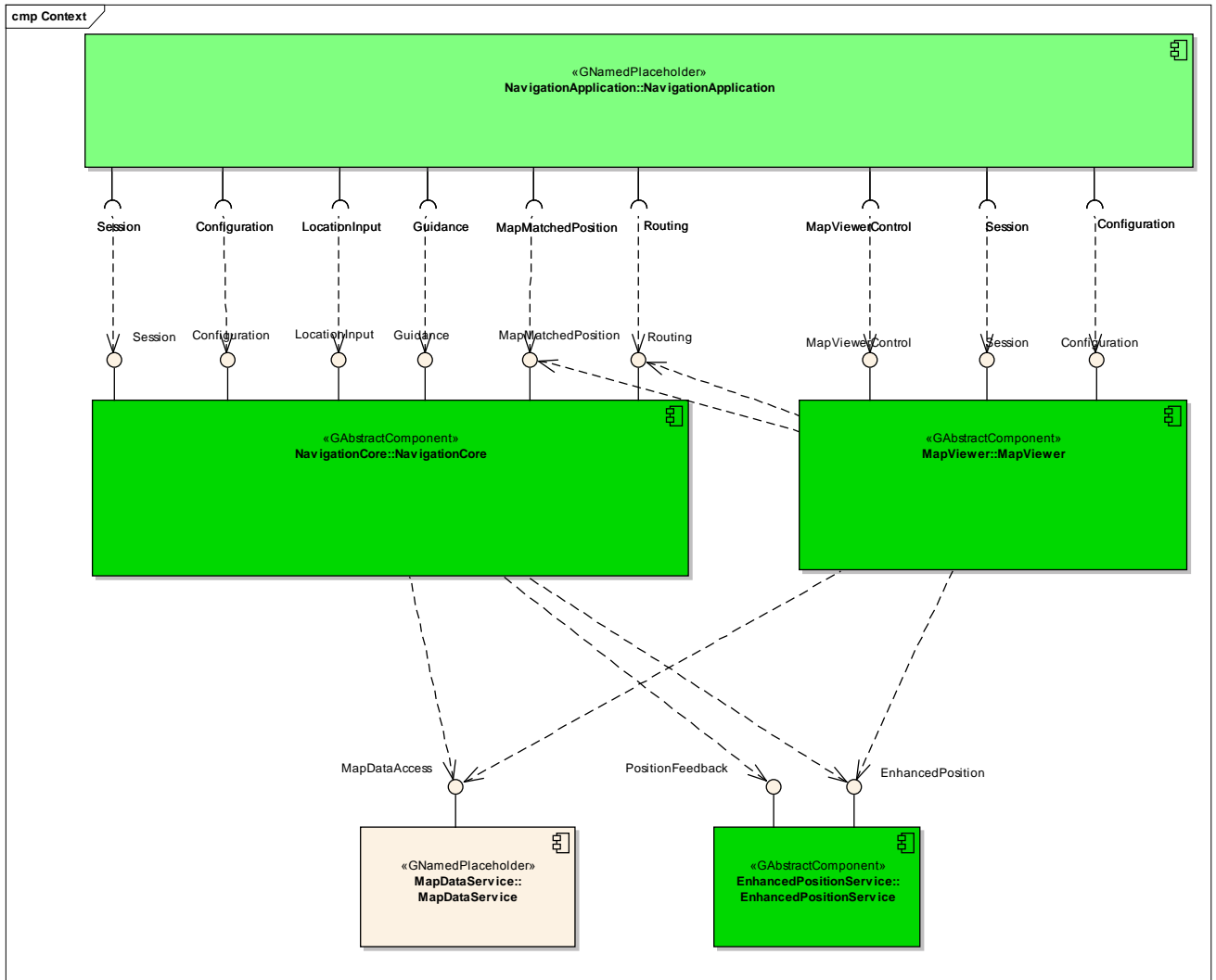
- Ease of Development
- Extensibility
- Multi-client Behavior
- Simplicity

5 Architecture

5.1 Interfaces



5.2 Interaction with other Components



6 API

6.1 D-Bus

The NavigationCore 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>

6.2 Git Repository

The NavigationCore interfaces can be found in the GENIVI Git repository at:
<https://git.genivi.org/git/gitweb.cgi?p=navigation;a=tree;f=NavigationCore/api>

6.3 Naming Conventions

Element	Description	Example
Interface File	genivi.<component name in lowercase character>.<interface name in lowercase characters>	genivi.navigationcore.routing.xml
Methods/Signal/Properties	Camel case naming convention First letter uppercase	CalculateRoute
Arguments	Camel case naming convention First letter lowercase	routeHandle

6.4 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

6.5 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.

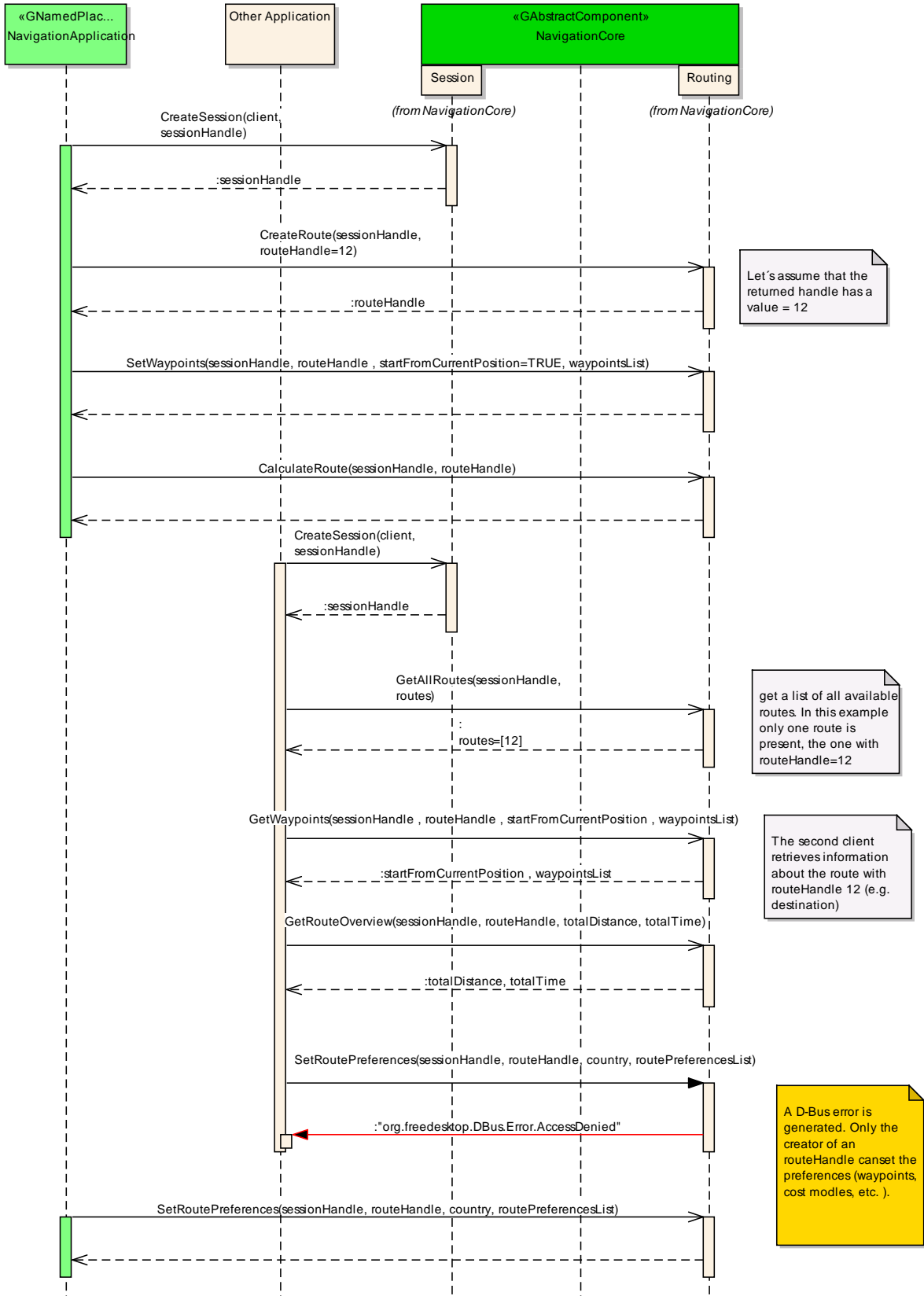
6.6 Sessions

A session-id identifies a requester. In a multi-client context only the requester that created an instance may be allowed to execute operations on that instance.

Other requesters may operate on the same handle (e.g. route handle, location input handle) with limited rights. For example, only the creator of a route handle can start/stop a route calculation on that handle. Other clients may simply be allowed to retrieve the total distance to the destination.

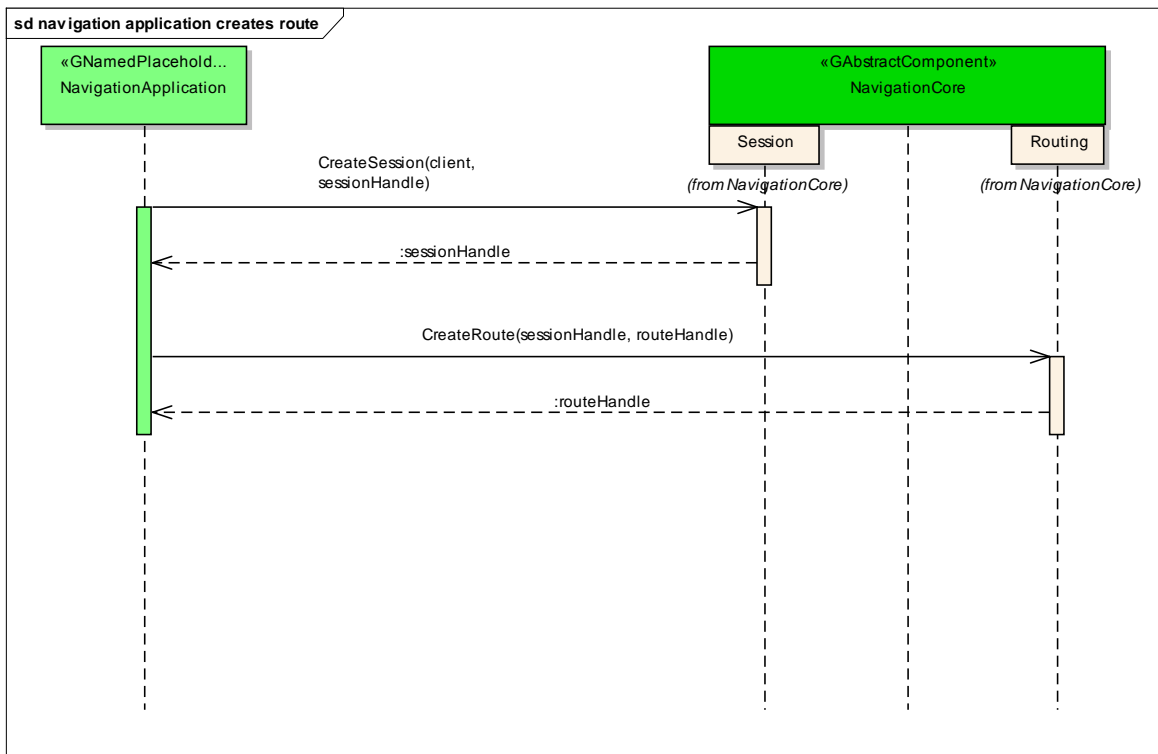
The following diagram shows an example of how session handles can be used:

sd two clients try to change route preferences of the same route

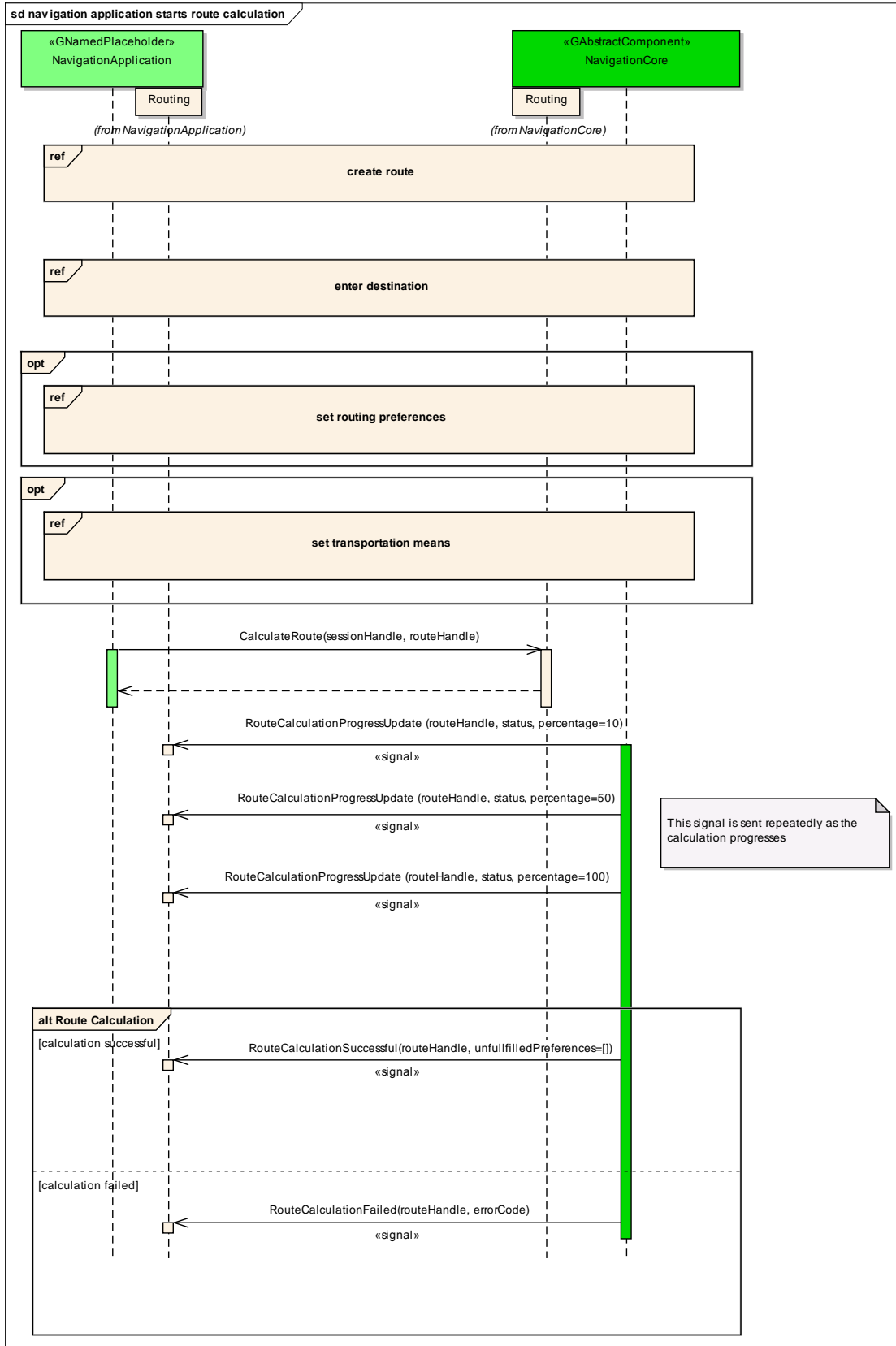


6.7 Sequence Diagrams

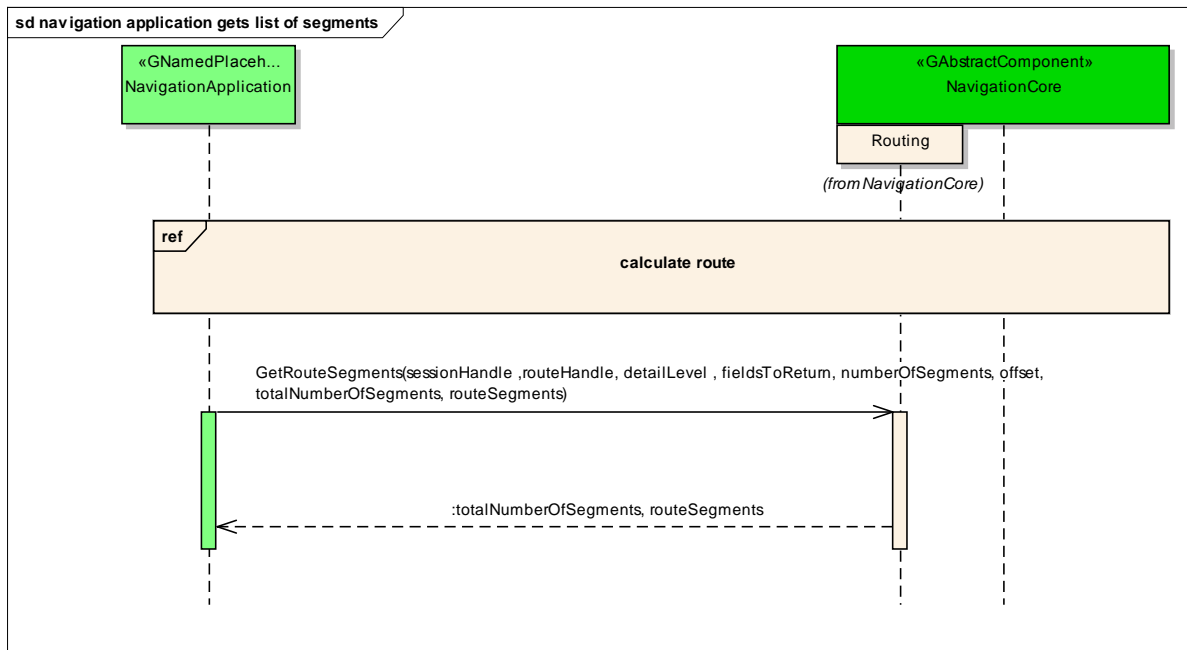
6.7.1 navigation application creates route



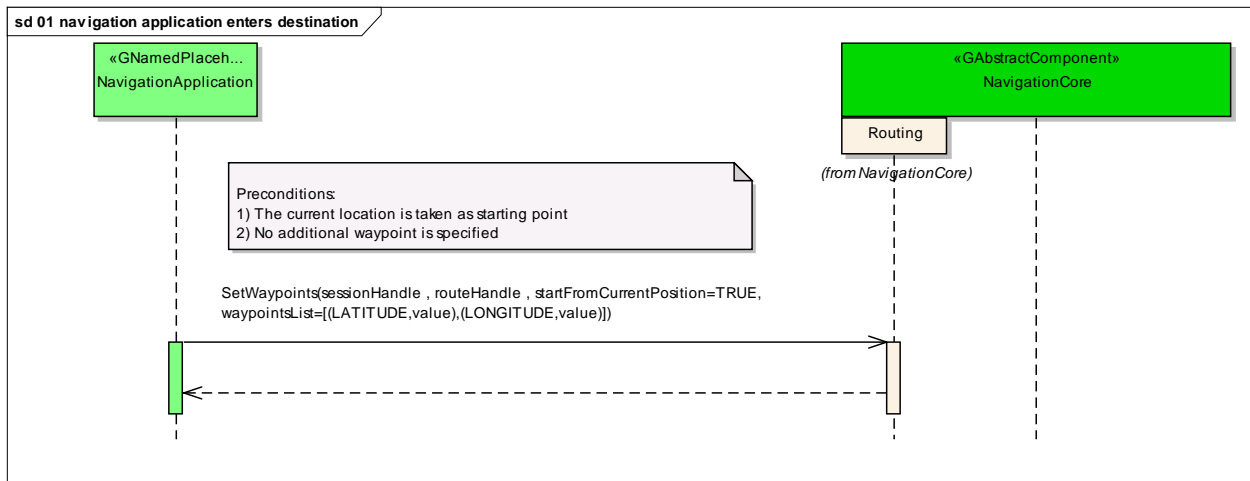
6.7.2 navigation application starts route calculation



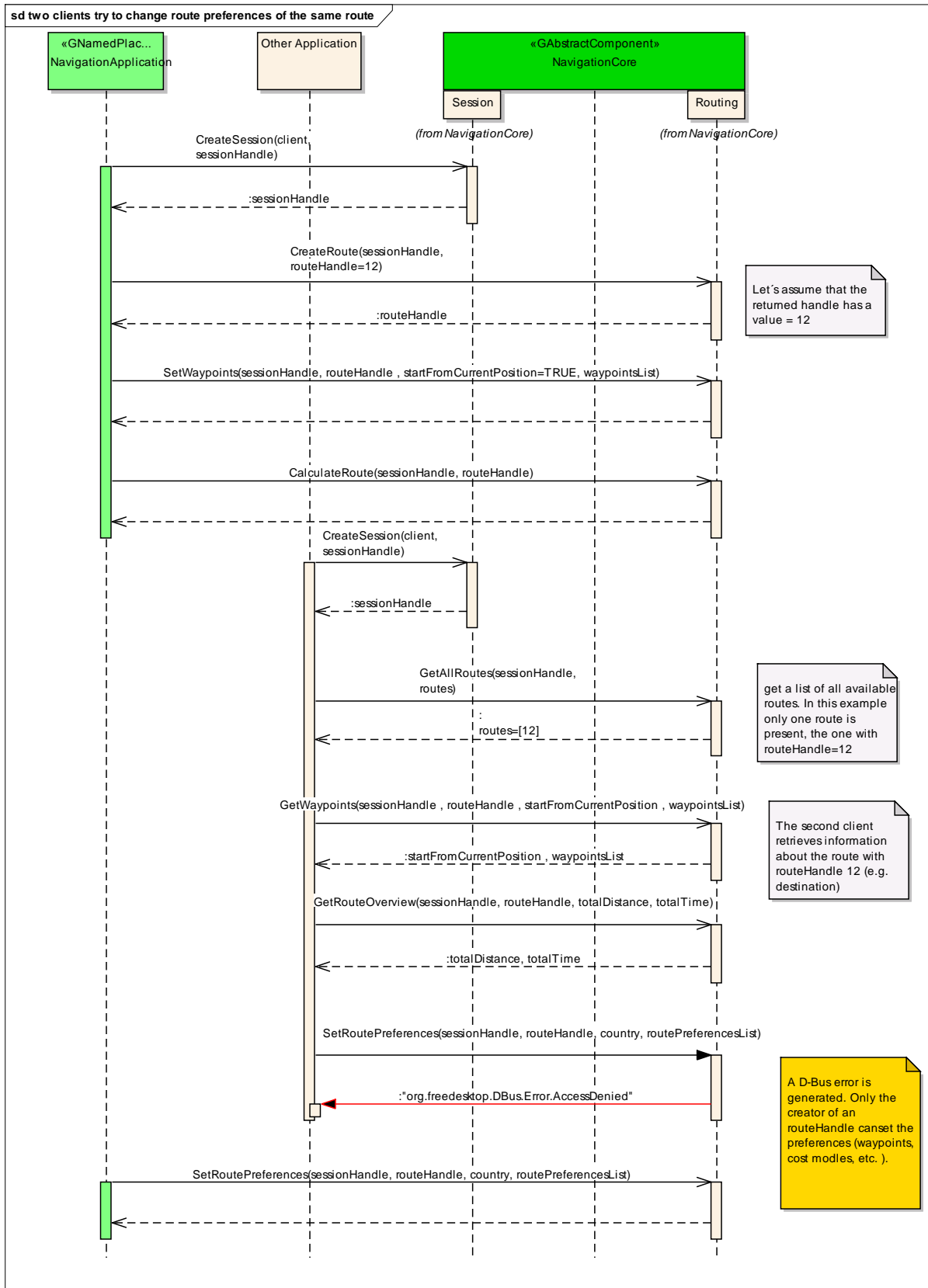
6.7.3 navigation application gets list of segments



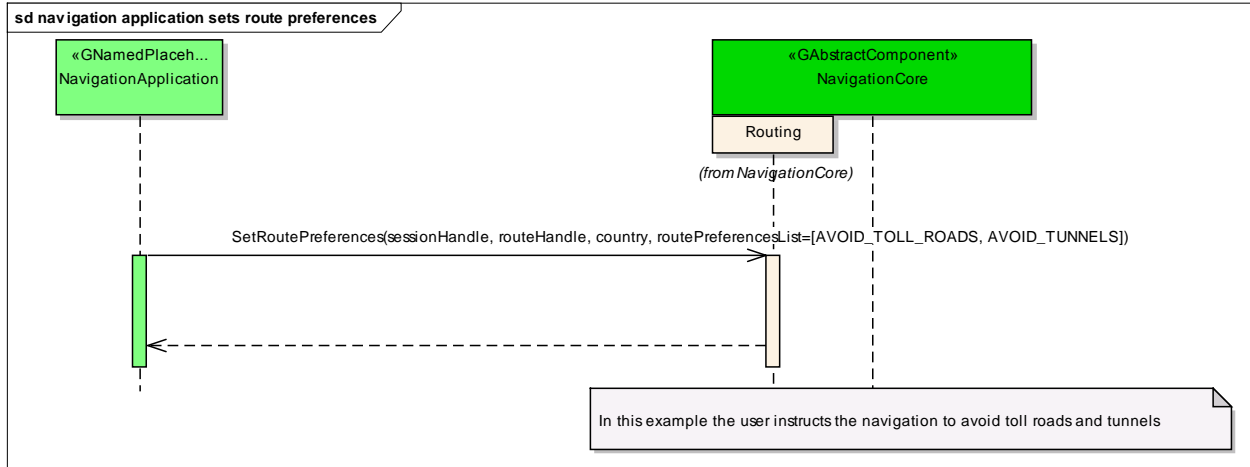
6.7.4 navigation application enters destination



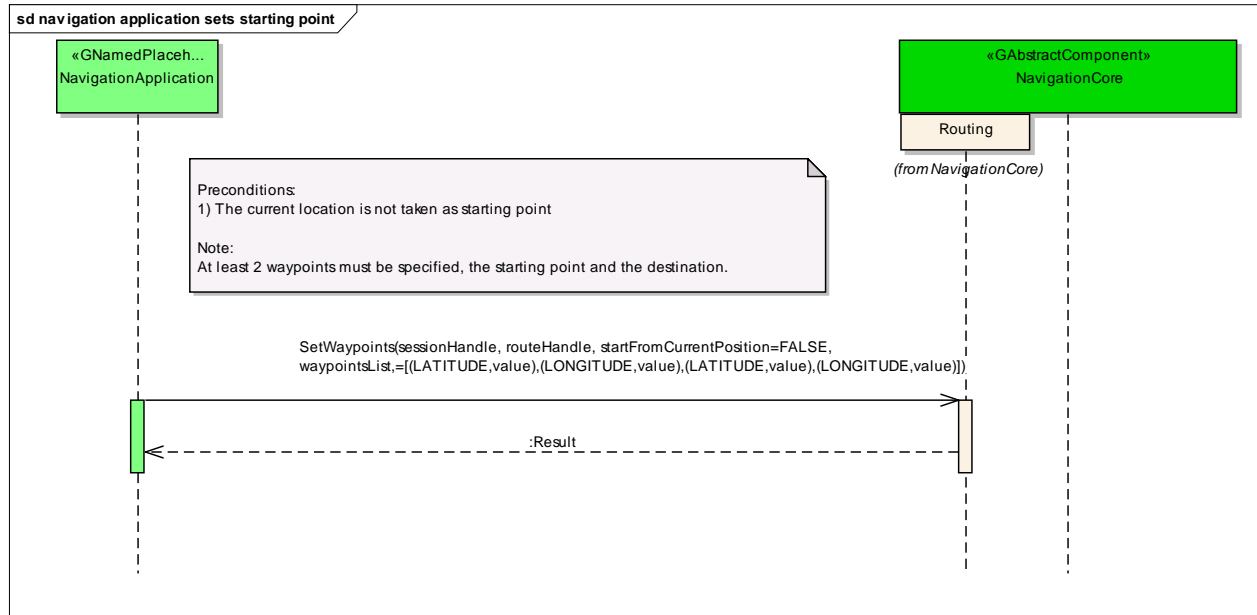
6.7.5 two clients try to change route preferences of the same route



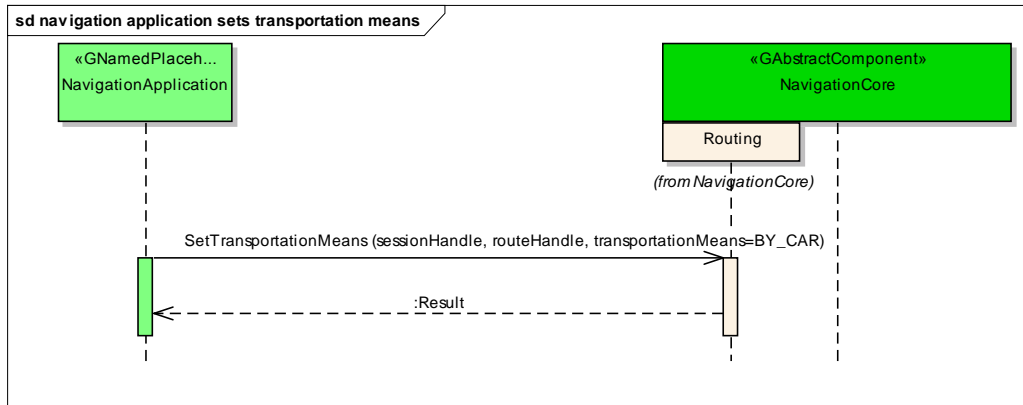
6.7.6 navigation application sets route preferences



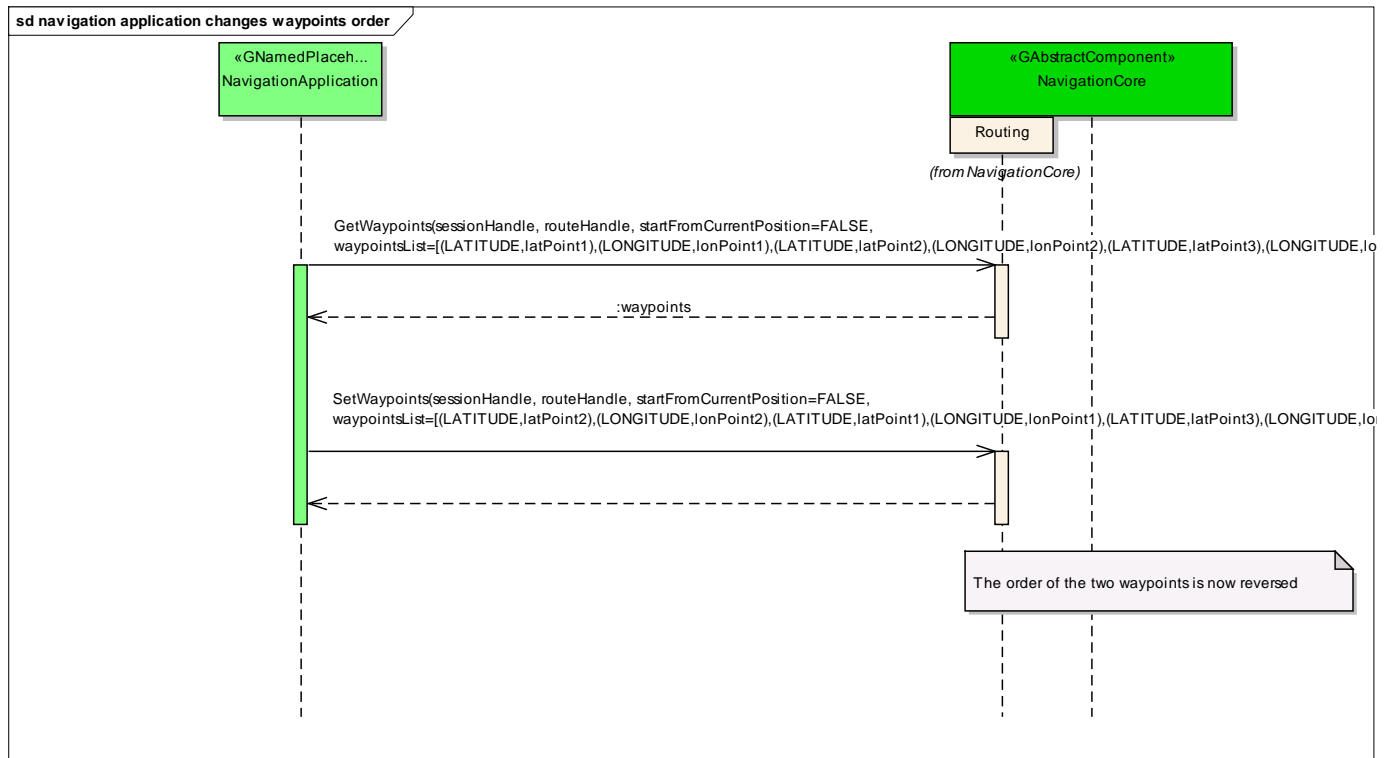
6.7.7 navigation application sets starting point



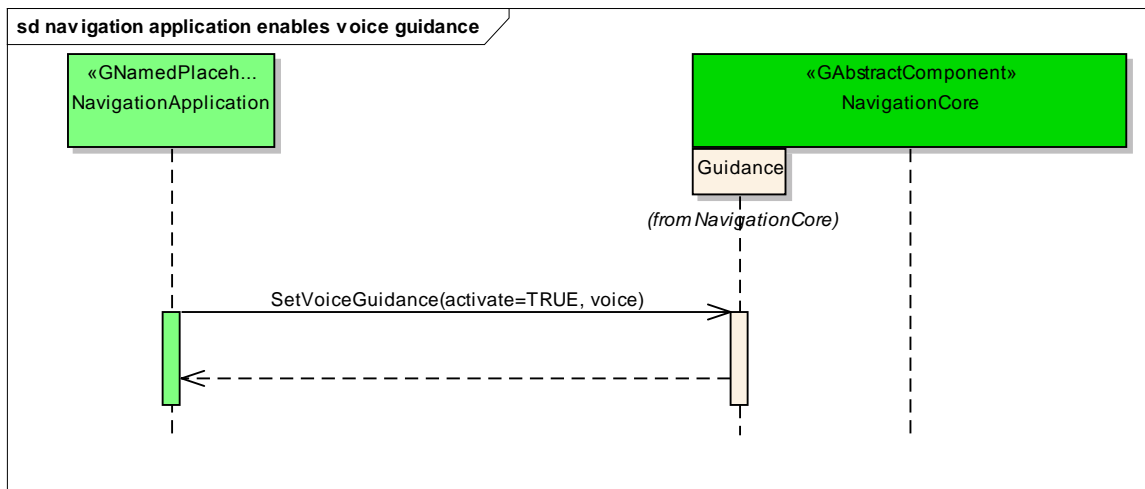
6.7.8 navigation application sets transportation means



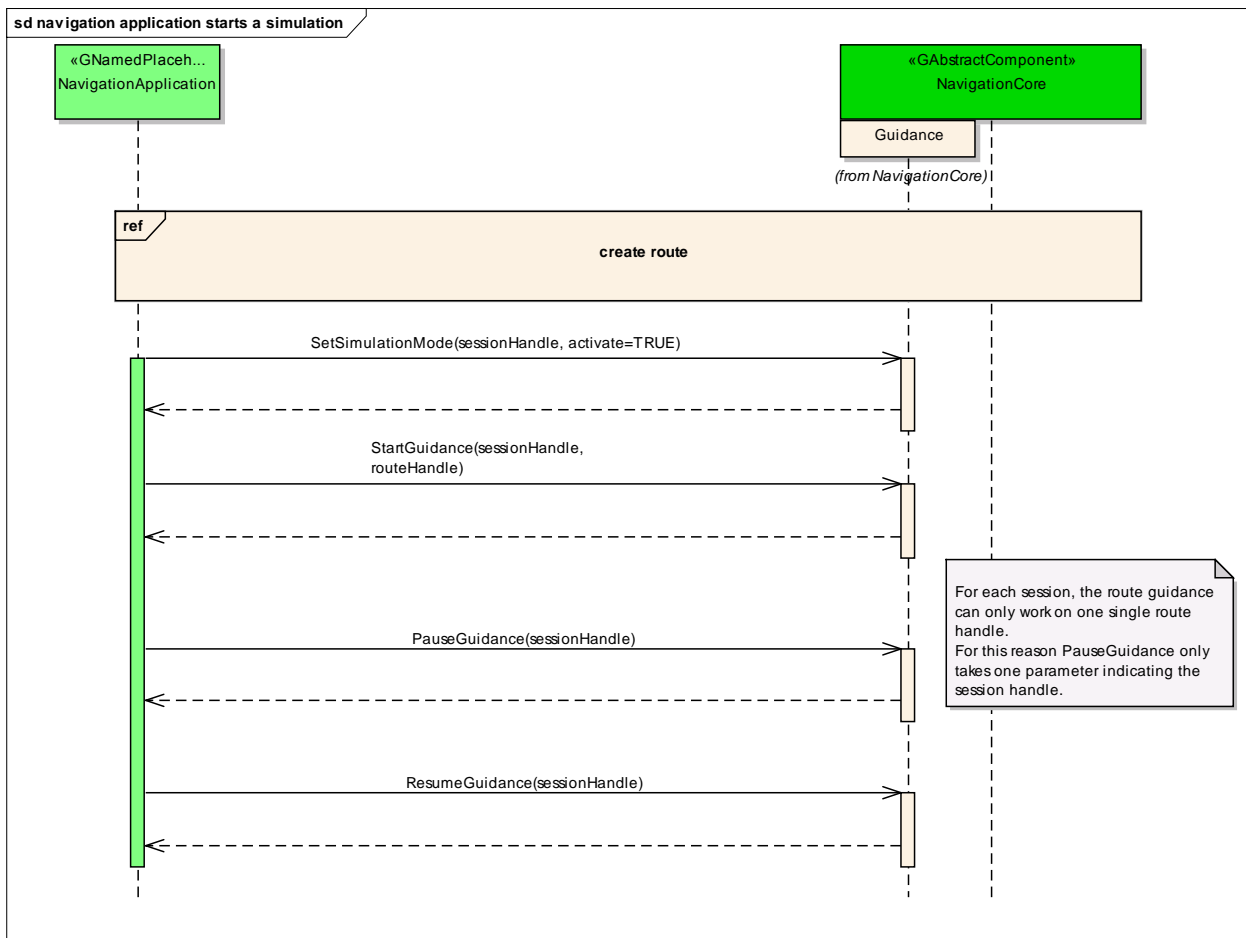
6.7.9 navigation application changes waypoints order



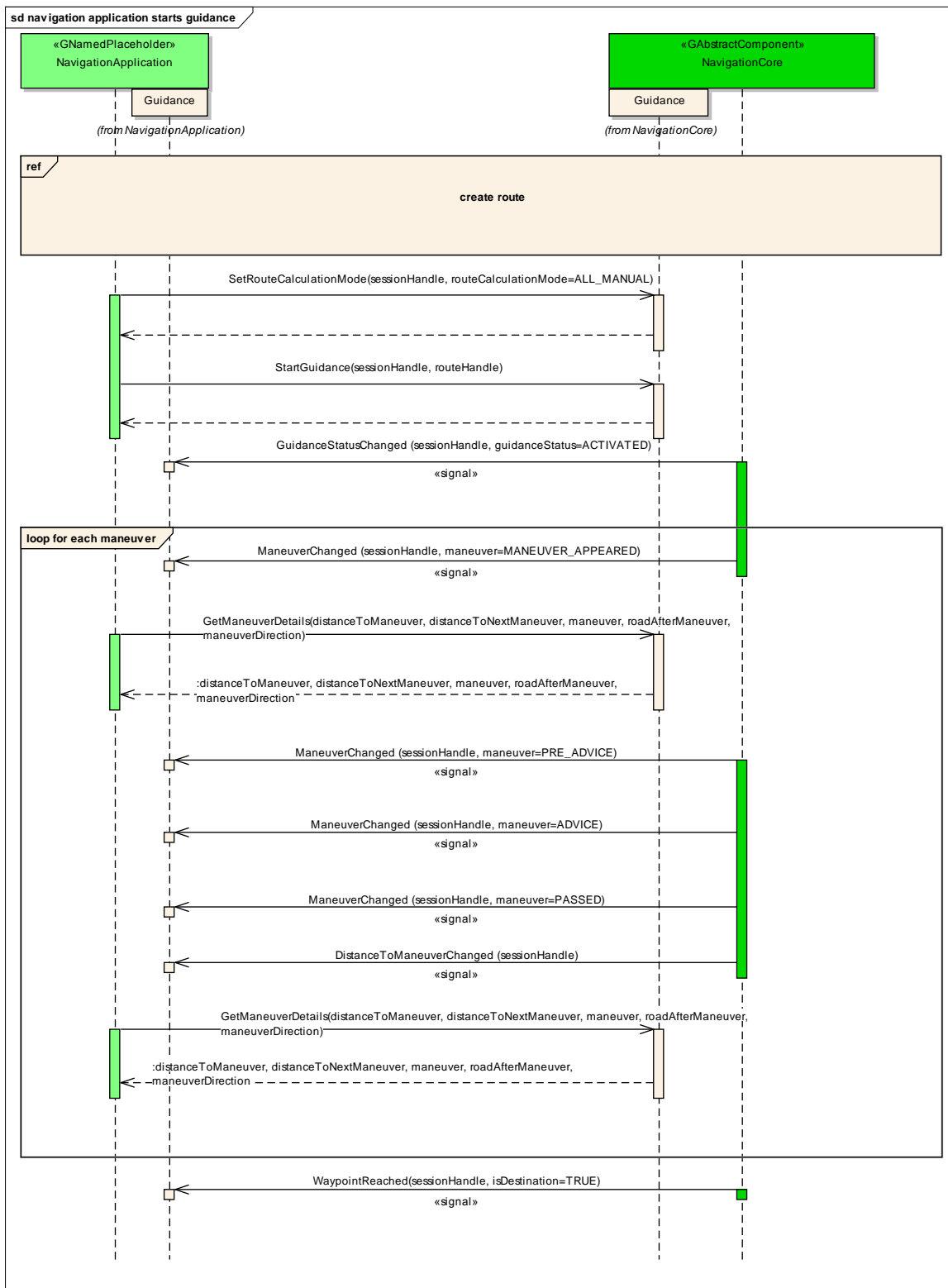
6.7.10 navigation application enables voice guidance



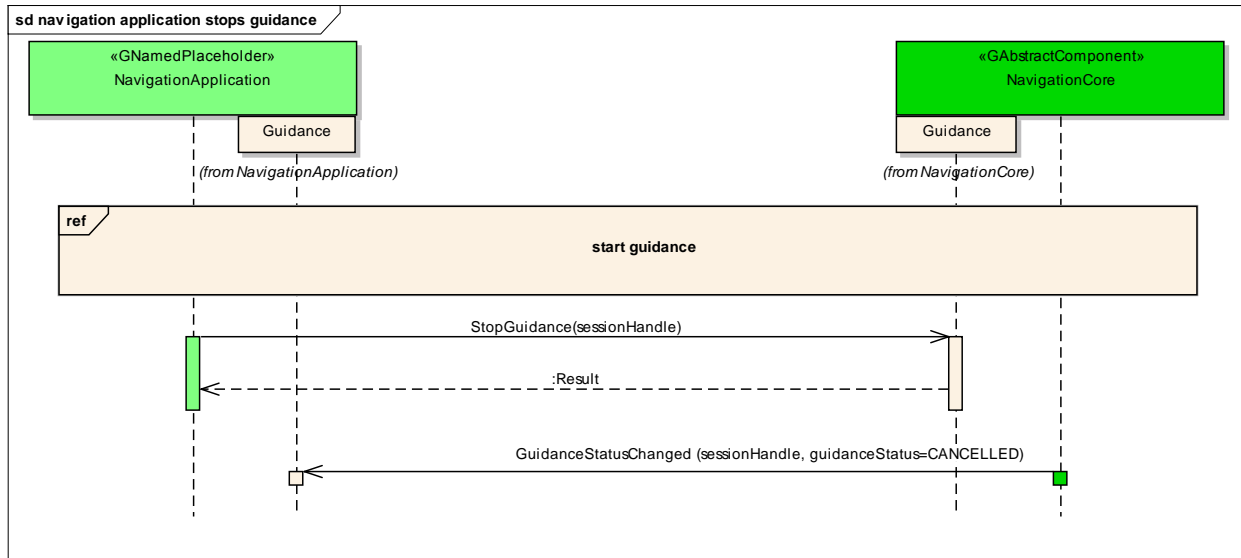
6.7.11 navigation application starts a simulation



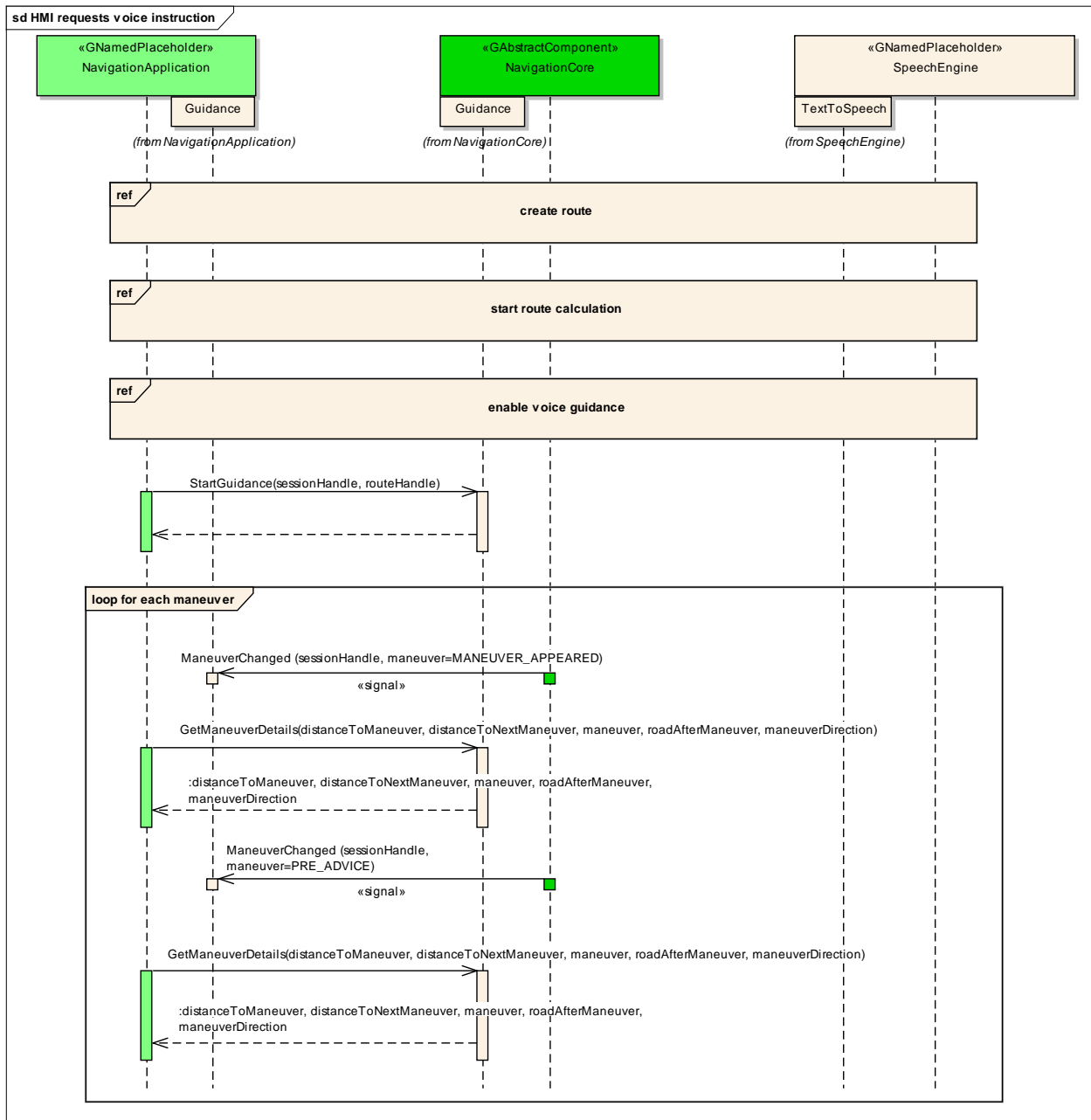
6.7.12 navigation application starts guidance



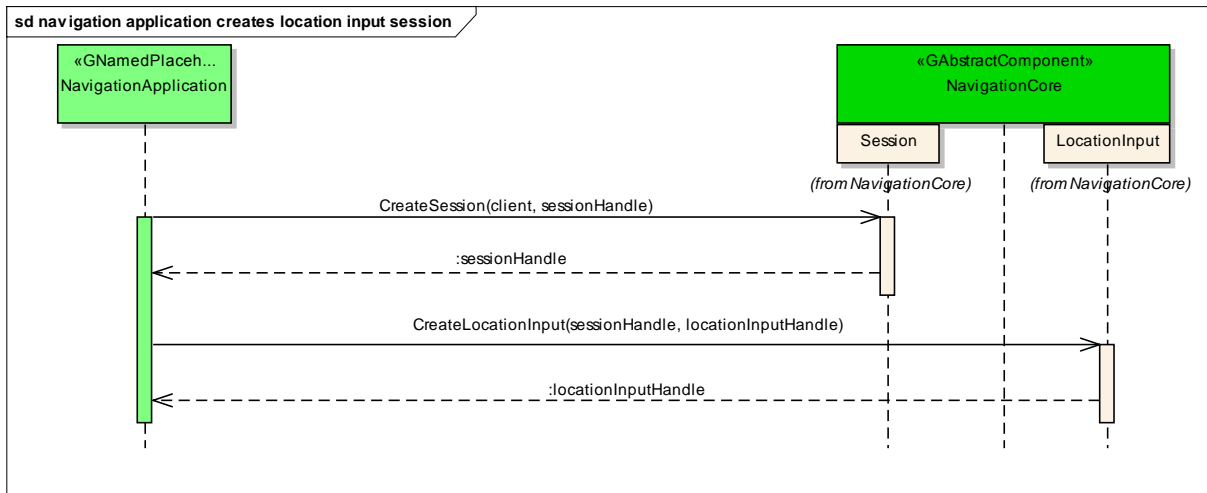
6.7.13 navigation application stops guidance



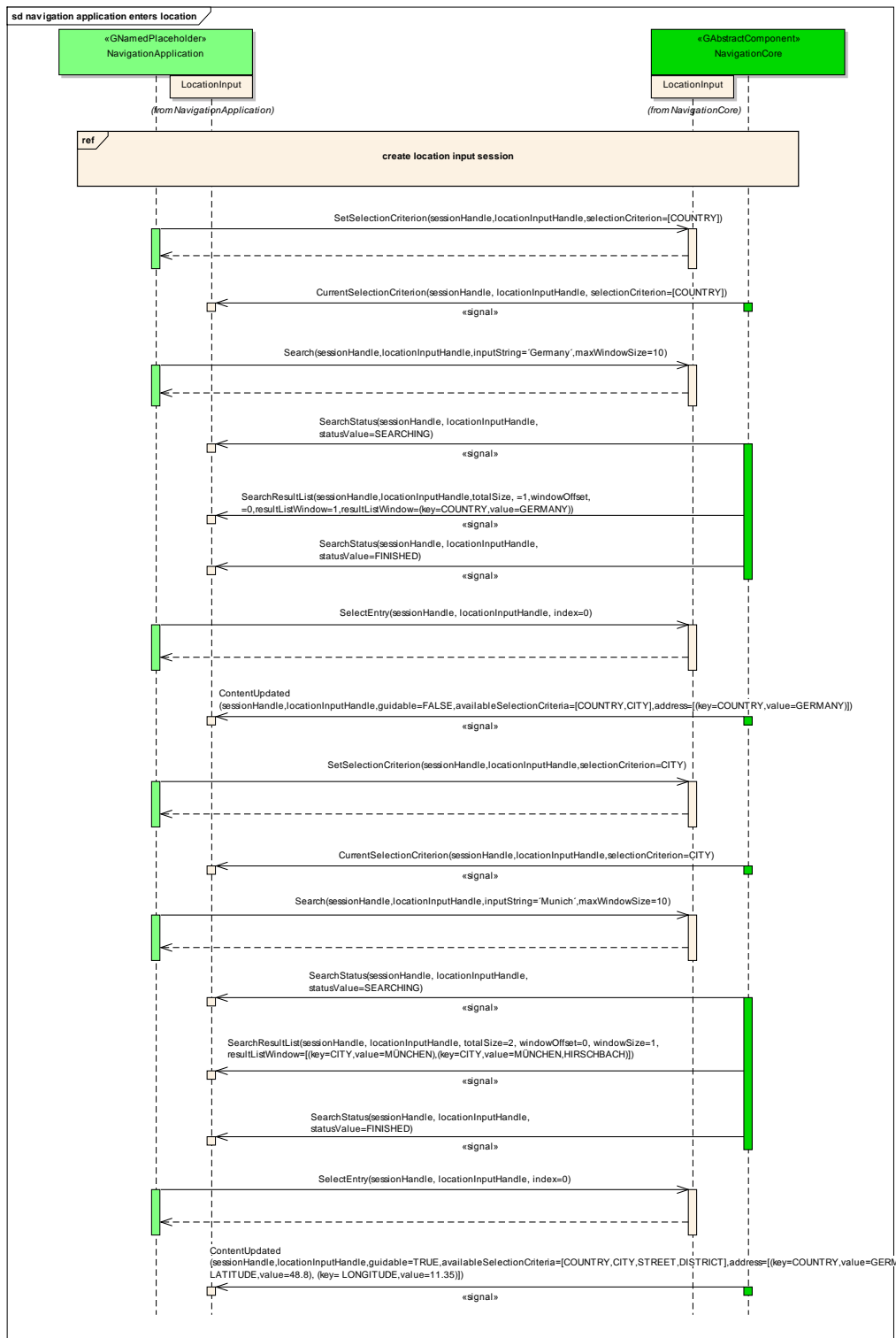
6.7.14 HMI requests voice instruction



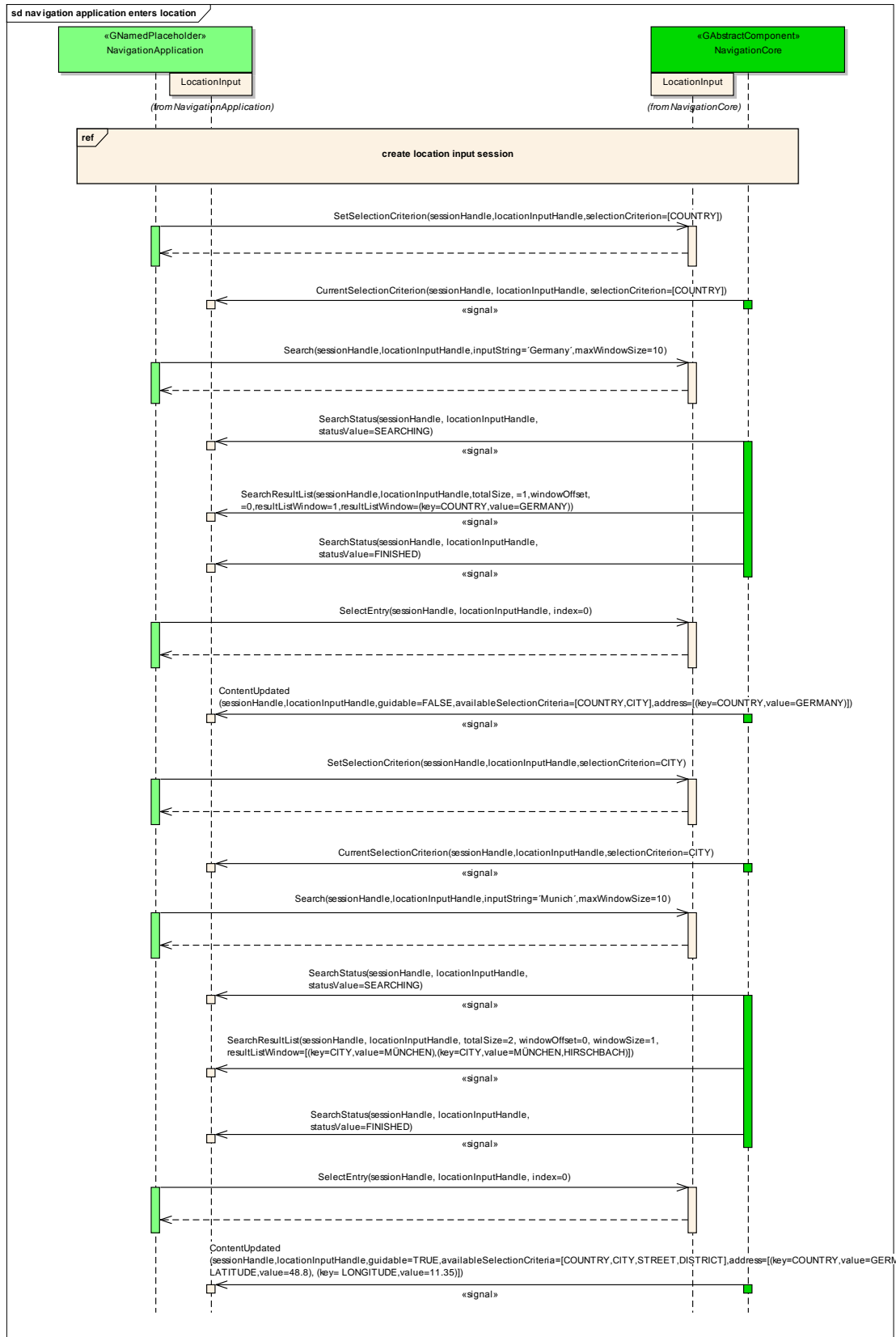
6.7.15 navigation application creates location input session



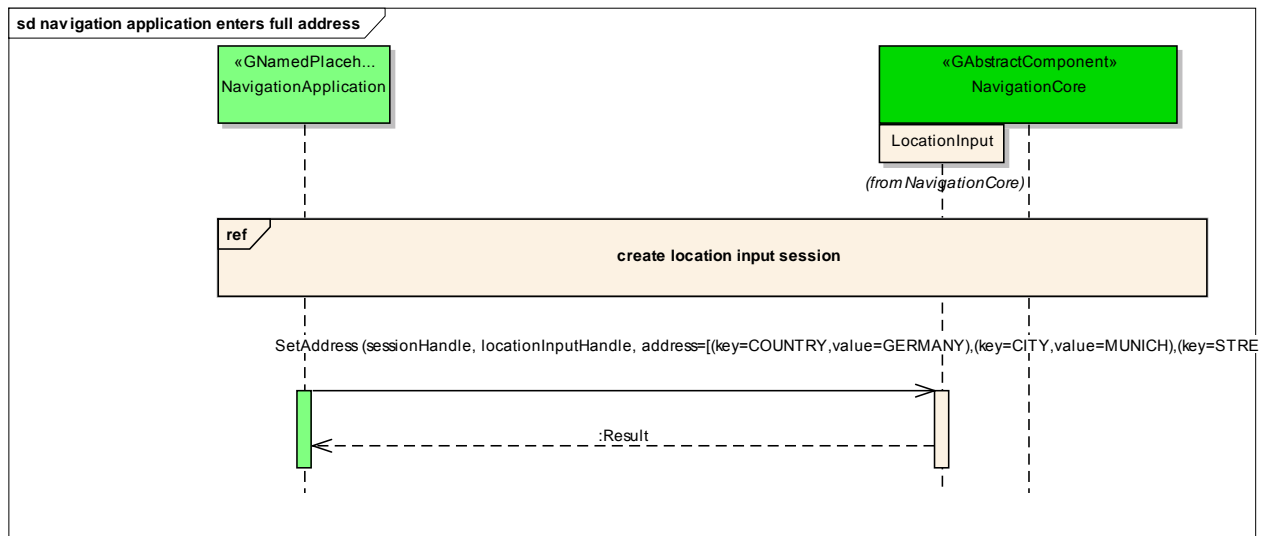
6.7.16 navigation application enters location



6.7.17 navigation application enters location using speller



6.7.18 navigation application enters full address



6.8 Interfaces

interface

org.genivi.navigationcore.Routing

version 3.0.0 (22-01-2014)

Routing = This interface offers functions that implement the routing functionality of a navigation system

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

CreateRoute = This method creates a route

method CreateRoute

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

out u routeHandle

This error is generated if no more routing handles are available

error org.genivi.navigationcore.Routing.Error.NoMoreRouteHandles

DeleteRoute = This method deletes a route and its associated resources

method DeleteRoute

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

This error is generated if an application tries to delete a route handle that is not available

error org.genivi.navigationcore.Routing.Error.RouteNotAvailable

This error is generated if an application tries to delete a route which is not created manually (e.g. an alternative route calculated in the background)

error org.genivi.navigationcore.Routing.Error.OperationNotAllowed

RouteDeleted = This signal is emitted to inform clients that the current route has been deleted

signal RouteDeleted

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value
out u routeHandle

SetCostModel = This method sets the cost model

method SetCostModel

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value
in u routeHandle

*costModel =
enum(INVALID,FASTEST,SHORTEST,ECOLOGICAL,SCENIC,EASY,OFF_ROAD,BALANCED,CHEAPEST, ...)
FASTEST = least time to reach the destination
SHORTEST = shortest distance to destination
ECOLOGICAL = least fuel or electric charge to destination
SCENIC = most scenic route to destination
EASY = least number of turns to reach the destination
OFF_ROAD = a distance-optimised route between points that are not covered by road mappings
BALANCED = trade-off between FASTEST and SHORTEST (e.g. 50% FASTEST and 50% SHORTEST)
CHEAPEST = least fuel or electric charge to destination taking pricing into account*
in q costModel

This error is generated if an application tries to set a cost model for a route which is not created manually (e.g. an alternative route calculated in the background)

error org.genivi.navigationcore.Routing.Error.OperationNotAllowed

GetCostModel = This method retrieves the selected cost model

method GetCostModel

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value
in u routeHandle

*costModel =
enum(INVALID,FASTEST,SHORTEST,ECOLOGICAL,SCENIC,EASY,OFF_ROAD,BALANCED,CHEAPEST, ...)
FASTEST = least time to reach the destination
SHORTEST = shortest distance to destination
ECOLOGICAL = least fuel or electric charge to destination
SCENIC = most scenic route to destination
EASY = least number of turns to reach the destination
OFF_ROAD = a distance-optimised route between points that are not covered by road mappings
BALANCED = trade-off between FASTEST and SHORTEST (e.g. 50% FASTEST and 50% SHORTEST)
CHEAPEST = least fuel or electric charge to destination taking pricing into account*
out q costModel

GetSupportedCostModels = This method retrieves a list of supported cost models

method GetSupportedCostModels

*costModelsList = array[costModel]
costModel =
enum(INVALID,FASTEST,SHORTEST,ECOLOGICAL,SCENIC,EASY,OFF_ROAD,BALANCED,CHEAPEST, ...)
FASTEST = least time to reach the destination
SHORTEST = shortest distance to destination
ECOLOGICAL = least fuel or electric charge to destination*

SCENIC = most scenic route to destination
EASY = least number of turns to reach the destination
OFF_ROAD = a distance-optimised route between points that are not covered by road mappings
BALANCED = trade-off between FASTEST and SHORTEST (e.g. 50% FASTEST and 50% SHORTEST)
CHEAPEST = least fuel or electric charge to destination taking pricing into account
out aq costModelsList

SetRoutePreferences = This method sets a list of route preferences

method SetRoutePreferences

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

countryCode = ISO 3166-1 alpha 3 country code (upper case)

If this argument is an empty string, it means that the preferences apply to all countries

in s countryCode

roadPreferenceList = array[preference]

preference = struct(preferenceMode,preferenceSource)

preferenceMode = enum(PROHIBIT,AVOID,USE,PREFER)

preferenceMode = PROHIBIT. Routing module must not calculate a planned route including sections matching given avoidance source.

preferenceMode = AVOID. Routing module should not calculate a planned route including sections matching given avoidance source.

preferenceMode = USE. Routing module should calculate a planned route including sections matching given avoidance source.

preferenceMode = PREFER. Routing module should calculate a planned route including as much as possible sections matching given avoidance source.

preferenceSource =

enum(FERRY,TOLL_ROADS,TUNNELS,HIGHWAYS_MOTORWAYS,VEHICLE_SIZE_LIMIT,CRIME_AREAS)

in a(qq) roadPreferenceList

conditionPreferenceList = array[preference]

preference = struct(preferenceMode,preferenceSource)

preferenceMode = enum(USE,IGNORE)

preferenceSource = enum(TRAFFIC_REALTIME, ...)

in a(qq) conditionPreferenceList

This error is generated if an application tries to set route preferences for a route which is not created manually (e.g. an alternative route calculated in the background)

error org.genivi.navigationcore.Routing.Error.OperationNotAllowed

This error is generated if an application tries to set a route preference which is not supported (e.g. (USE,CRIME_AREA)). The preferences which are not supported are product dependent.

error org.genivi.navigationcore.Routing.Error.RoutePreferenceNotSupported

GetRoutePreferences = This method retrieves a list of selected route preferences

method GetRoutePreferences

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

countryCode = ISO 3166-1 alpha 3 country code (upper case)

If this argument is an empty string, it means that the preferences apply to all countries

in s countryCode

```
roadPreferenceList = array[preference]
preference = struct(preferenceMode,preferenceSource)
preferenceMode = enum(PROHIBIT,AVOID,USE,PREFER)
preferenceMode = PROHIBIT. Routing module must not calculate a planned route including sections matching given avoidance source.
preferenceMode = AVOID. Routing module should not calculate a planned route including sections matching given avoidance source.
preferenceMode = USE. Routing module should calculate a planned route including sections matching given avoidance source.
preferenceMode = PREFER. Routing module should calculate a planned route including as much as possible sections matching given avoidance source.
preferenceSource =
enum(FERRY,TOLL_ROADS,TUNNELS,HIGHWAYS_MOTORWAYS,VEHICLE_SIZE_LIMIT,CRIME_AREAS)
Out a(qq) roadPreferenceList
```

```
conditionPreferenceList = array[preference]
preference = struct(preferenceMode,preferenceSource)
preferenceMode = enum(USE,IGNORE)
preferenceSource = enum(TRAFFIC_REALTIME, ...)
Out a(qq) conditionPreferenceList
```

GetSupportedRoutePreferences = This method retrieves a list of supported route preferences

method GetSupportedRoutePreferences

```
routePreferencesList = array[preference]
preference = struct(preferenceMode,preferenceSource)
preferenceMode = enum(PROHIBIT,AVOID,USE,PREFER)
preferenceMode = PROHIBIT. Routing module must not calculate a planned route including sections matching given avoidance source.
preferenceMode = AVOID. Routing module should not calculate a planned route including sections matching given avoidance source.
preferenceMode = USE. Routing module should calculate a planned route including sections matching given avoidance source.
preferenceMode = PREFER. Routing module should calculate a planned route including as much as possible sections matching given avoidance source.
preferenceSource =
enum(FERRY,TOLL_ROADS,TUNNELS,HIGHWAYS_MOTORWAYS,VEHICLE_SIZE_LIMIT,CRIME_AREAS)
Out a(qq) routePreferencesList
```

```
conditionPreferenceList = array[preference]
preference = struct(preferenceMode,preferenceSource)
preferenceMode = enum(USE,IGNORE)
preferenceSource = enum(TRAFFIC_REALTIME, ...)
Out a(qq) conditionPreferenceList
```

SetRouteSchedule = This method sets the time schedule for the route to be calculated

method SetRouteSchedule

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

routeSchedule = array[detail]

detail = dictionary[key,value]
dictionary = array of tuples (key,value)
key = enum(ARRIVAL_TIME, ARRIVAL_DATE,DEPARTURE_TIME, DEPARTURE_DATE)
key = ARRIVAL_TIME, value of type 'u', that represents the arrival time is expressed in seconds since mid-night (UTC)
key = ARRIVAL_DATE, value of type 'u', that represents the arrival date is either a calendar date (the number of days since 1 Jan 2000) or a weekday indication. For weekday indication the values 0 till 6 are used (0 = Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday)
key = DEPARTURE_TIME, value of type 'u', that represents the departure time is expressed in seconds since mid-night (UTC)
key = DEPARTURE_DATE, value of type 'u', that represents the departure date is either a calendar date (the number of days since 1 Jan 2000) or a weekday indication. For weekday indication the values 0 till 6 are used (0 = Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday)
in a{qu} routeSchedule

GetRouteSchedule = This method gets the time schedule for the route to be calculated

method GetRouteSchedule

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

valuesToReturn= array[value]

value= enum(ARRIVAL_TIME, ARRIVAL_DATE,DEPARTURE_TIME, DEPARTURE_DATE)

in aq valuesToReturn

routeSchedule = array[detail]

detail = dictionary[key,value]

dictionary = array of tuples (key,value)

key = enum(ARRIVAL_TIME, ARRIVAL_DATE,DEPARTURE_TIME, DEPARTURE_DATE)

key = ARRIVAL_TIME, value of type 'u', that represents the arrival time is expressed in seconds since mid-night (UTC)

key = ARRIVAL_DATE, value of type 'u', that represents the arrival date is either a calendar date (the number of days since 1 Jan 2000) or a weekday indication. For weekday indication the values 0 till 6 are used (0 = Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday)

key = DEPARTURE_TIME, value of type 'u', that represents the departure time is expressed in seconds since mid-night (UTC)

key = DEPARTURE_DATE, value of type 'u', that represents the departure date is either a calendar date (the number of days since 1 Jan 2000) or a weekday indication. For weekday indication the values 0 till 6 are used (0 = Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday)

out a{qu} routeSchedule

SetTransportationMeans = This method sets a list of means of transportation that must be considered when calculating a route

method SetTransportationMeans

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

transportationMeansList = array[transportationMeans]

transportationMeans =

enum(INVALID,BY_CAR,ON_FOOT,LONG_RANGE_TRAINS,PUBLIC_TRANSPORTATION,BY_BICYCLE,BY_TRUCK, ...)

in aq transportationMeansList

This error is generated if an application tries to set transportation means for a route which is not created manually (e.g. an alternative route calculated in the background)

error `org.genivi.navigationcore.Routing.Error.OperationNotAllowed`

GetTransportationMeans = This method retrieves the selected means of transportation

method `GetTransportationMeans`

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u `routeHandle`

transportationMeansList = array[transportationMeans]

transportationMeans =

enum(INVALID,BY_CAR,ON_FOOT,LONG_RANGE_TRAINS,PUBLIC_TRANSPORTATION,BY_BICYCLE,BY_TRUCK, ...)

out aq `transportationMeansList`

GetSupportedTransportationMeans = This method retrieves a list of supported means of transportation

method `GetSupportedTransportationMeans`

transportationMeansList = array[transportationMeans]

transportationMeans =

enum(INVALID,BY_CAR,ON_FOOT,LONG_RANGE_TRAINS,PUBLIC_TRANSPORTATION,BY_BICYCLE,BY_TRUCK, ...)

out aq `transportationMeansList`

SetExcludedAreas = This method sets the areas to be excluded when calculating a route

method `SetExcludedAreas`

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u `sessionHandle`

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u `routeHandle`

excludedAreas = array[array[lat,lon]]

excludedAreas = array[convexPolygon]

convexPolygon = array[lat,lon]

lat = latitude of a vertex of the polygon in format %3.6f. Range [-90:+90]. Example: 48.053250

lon = longitude of a vertex of the polygon in format %3.6f. Range [-180:+180]. Example: 48.053250

Note: a polygon must have at least 3 vertexes

in aa(dd) `excludedAreas`

This error is generated if an application tries to set excluded areas for a route which is not created manually (e.g. an alternative route calculated in the background)

error `org.genivi.navigationcore.Routing.Error.OperationNotAllowed`

GetExcludedAreas = This method retrieves the areas to be excluded when calculating a route

method `GetExcludedAreas`

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u `routeHandle`

excludedAreas = array[array(lat,lon)]
excludedAreas = array[convexPolygon]
convexPolygon = array[lat,lon]
lat = latitude of a vertex of the polygon in format %3.6f. Range [-90:+90]. Example: 48.053250
lon = longitude of a vertex of the polygon in format %3.6f. Range [-180:+180]. Example: 48.053250
Note: pass an empty array to remove previously selected excluded areas
out aa(dd) excludedAreas

SetWaypoints = This method sets a list of waypoints

method SetWaypoints

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u routeHandle

startFromCurrentPosition = flag indicating if the current position is used as starting point
in b startFromCurrentPosition

waypointsList = array[waypoint]
waypoint = tuple (key,value)
key = enum(INVALID,WAYPOINT_TYPE,LOCATION_INPUT,LATITUDE,LONGITUDE,ALTITUDE, ...)
key = WAYPOINT_TYPE, value = value of type 'q', that represents an enum(INVALID,SOFT_POINT,HARD_POINT, ...)
key = LOCATION_INPUT, value = value of type 'ay'. This is a byte array whose interpretation is left to the navigation core
key = LATITUDE, value = value of type 'd', that expresses the latitude of the starting point in format %3.6f. Range [-90:+90]. Example: 48.053250
key = LONGITUDE, value = value of type 'd', that expresses the longitude of the starting point in format %3.6f. Range [-180:+180]. Example: 8.324500
key = ALTITUDE, value = value of type 'i', that expresses the altitude of the starting point in meters
Note: if the flag StartFromCurrentPosition=true, then at least one waypoint must be provided (destination)
Note: if the flag StartFromCurrentPosition=false, then at least two waypoints must be provided (starting point and destination)
in aa{qv} waypointsList

This error is sent when a client application tries to change the waypoints while the route guidance is active
error org.genivi.navigationcore.Routing.Error.WaypointCannotBeChanged

This error is sent when a client application tries to set a number of waypoints that exceeds the system capabilities
error org.genivi.navigationcore.Routing.Error.TooManyWaypoints

This error is generated if an application tries to set waypoints for a route which is not created manually (e.g. an alternative route calculated in the background)
error org.genivi.navigationcore.Routing.Error.OperationNotAllowed

GetWaypoints = This method retrieves a list of waypoints

method GetWaypoints

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u routeHandle

startFromCurrentPosition = flag indicating if the current position is used as starting point
out b startFromCurrentPosition

waypointsList = array[waypoint]

waypoint = tuple (key,value)
key = enum(INVALID,WAYPOINT_TYPE,LOCATION_INPUT,LATITUDE,LONGITUDE,ALTITUDE, ...)
key = WAYPOINT_TYPE, value = value of type 'q', that represents an enum(INVALID,SOFT_POINT,HARD_POINT, ...)
key = LOCATION_INPUT, value = value of type 'ay'. This is a byte array whose interpretation is left to the navigation core
key = LATITUDE, value = value of type 'd', that expresses the latitude of the starting point in format %3.6f. Range [-90:+90]. Example: 48.053250
key = LONGITUDE, value = value of type 'd', that expresses the longitude of the starting point in format %3.6f. Range [-180:+180]. Example: 8.324500
key = ALTITUDE, value = value of type 'i', that expresses the altitude of the starting point in meters
Out **aa{qv}** waypointsList

CalculateRoute = This method starts a route calculation

method CalculateRoute

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u routeHandle

This error is generated if an application tries to calculate a route which is not created manually (e.g. an alternative route calculated in the background)

error *org.genivi.navigationcore.Routing.Error.OperationNotAllowed*

CancelRouteCalculation = This method cancels a route calculation

method CancelRouteCalculation

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u routeHandle

RouteCalculationCancelled = This signal informs a client that a route calculation was cancelled

signal RouteCalculationCancelled

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
out u routeHandle

RouteCalculationSuccessful = This signal informs a client that a route calculation was successful

signal RouteCalculationSuccessful

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
out u routeHandle

unfulfilledPreferences = array[preference]

preference = tuple(preferenceMode,preferenceSource)

preferenceMode = enum(PROHIBIT,AVOID,USE,PREFER)

preferenceMode = PROHIBIT. Routing module must not calculate a planned route including sections matching given avoidance source.

preferenceMode = AVOID. Routing module should not calculate a planned route including sections matching given avoidance source.

preferenceMode = USE. Routing module should calculate a planned route including sections matching given avoidance source.

preferenceMode = PREFER. Routing module should calculate a planned route including as much as possible sections matching given avoidance source.

preferenceSource =

enum(FERRY,TOLL_ROADS,TUNNELS,HIGHWAYS_MOTORWAYS,VEHICLE_SIZE_LIMIT,CRIME_AREAS)

Out a{qq} unfulfilledPreferences

RouteCalculationFailed = This signal informs a client that a route calculation failed

signal RouteCalculationFailed

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

out u routeHandle

errorCode =

enum(INVALID,UNMATCHED_POSITION,UNREACHABLE_DESTINATION,UNFULFILLED_PREFERENCE_MODE, ...)

errorCode = UNFULFILLED_PREFERENCE_MODE. Refer to unfulfilledPreferences to see which routing preferences could not be met causing routing calculation to fail; for any other value the argument unfulfilledPreferences should be ignored.

out q errorCode

unfulfilledPreferences = array[preference]

preference = tuple(preferenceMode,preferenceSource)

preferenceMode = enum(PROHIBIT,AVOID,USE,PREFER)

preferenceMode = PROHIBIT. Routing module must not calculate a planned route including sections matching given avoidance source.

preferenceMode = AVOID. Routing module should not calculate a planned route including sections matching given avoidance source.

preferenceMode = USE. Routing module should calculate a planned route including sections matching given avoidance source.

preferenceMode = PREFER. Routing module should calculate a planned route including as much as possible sections matching given avoidance source.

preferenceSource =

enum(FERRY,TOLL_ROADS,TUNNELS,HIGHWAYS_MOTORWAYS,VEHICLE_SIZE_LIMIT,CRIME_AREAS)

Out a{qq} unfulfilledPreferences

RouteCalculationProgressUpdate = This signal informs a client about a route calculation progress

signal RouteCalculationProgressUpdate

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

out u routeHandle

status = enum(INVALID,CALCULATION_OK,NO_POSITION, ...)

out q status

percentage = progress status. Range [0:100]

out y percentage

CalculateRoutes = This method allows a client to calculate alternative routes that differs from a list of already calculated routes

method CalculateRoutes

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

calculatedRoutesList = array[calculatedRoute]

calculatedRoute = Handle identifying an already calculated route. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in au calculatedRoutesList

alternativeRoutesList = array[alternativeRoute]

alternativeRoute = Handle identifying an alternative route. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

Out au alternativeRoutesList

This error is generated if an application tries to calculate an alternative to a route which is not created manually (e.g. an alternative route calculated in the background)

error `org.genivi.navigationcore.Routing.Error.OperationNotAllowed`

GetRouteSegments = This method retrieves a list of segments for a given route starting from the one closest to the current position to the one closest to the destination

method GetRouteSegments

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

detailLevel = detail level

in n detailLevel

valuesToReturn= array[key]

key = enum(INVALID, LINK-

ID, INTERMEDIATE_POINTS, START_LATITUDE, START_LONGITUDE, START_ALTITUDE,

END_LATITUDE, END_LONGITUDE, END_ALTITUDE, ROAD_NAME, ROAD_NUMBER, DISTANCE, TIME, MANEUVER, INSTRUCTION,

BORDER_CROSSING, TIME_ZONE, DAYLIGHT_SAVING_TIME, ADDITIONAL_INFORMATION, HIGHWAY_EXIT, ...

, ALL)

in aq valuesToReturn

numberOfSegments = number of segments to be retrieved

in u numberOfSegments

offset = offset from the beginning of the list

in u offset

totalNumberOfSegments = total number of segments

out u totalNumberOfSegments

routeSegments = array[segment]

segment = tuple(key, value)

Note: a segment is the shortest navigable stretch of a route (e.g. the stretch between two adjacent junctions)

key = enum(INVALID, LINK-

ID, INTERMEDIATE_POINTS, START_LATITUDE, START_LONGITUDE, START_ALTITUDE,

END_LATITUDE, END_LONGITUDE, END_ALTITUDE, ROAD_NAME, ROAD_NUMBER, DISTANCE, TIME, MANEUVER, INSTRUCTION,

BORDER_CROSSING, TIME_ZONE, DAYLIGHT_SAVING_TIME, ADDITIONAL_INFORMATION, HIGHWAY_EXIT, ...

)

key = LINK-ID, value = value of type 'ay', that represents a link-ID in a format whose interpretation is left to the navigationcore

key = START_LATITUDE, value = value of type 'd', that expresses the latitude of the starting point in format %3.6f.

Range [-90:+90]. Example: 48.053250

key = END_LATITUDE, value = value of type 'd', that expresses the latitude of the ending point in format %3.6f.

Range [-90:+90]. Example: 48.053250

Note: END_LATITUDE can be omitted, if it coincides with the latitude of the start point of the next segment

key = START_LONGITUDE, value = value of type 'd', that expresses the longitude of the starting point in format %3.6f. Range [-180:+180]. Example: 8.321000

key = END_LONGITUDE, value = value of type 'd', that expresses the longitude of the ending point in format %3.6f. Range [-180:+180]. Example: 8.321000

Note: END_LONGITUDE can be omitted, if it coincides with the longitude of the start point of the next segment

key = START_ALTITUDE, value = value of type 'i', that expresses the altitude relative to the ground of the starting point in meters

key = END_ALTITUDE, value = value of type 'i', that expresses the altitude relative to the ground of the ending point in meters

Note: END_ALTITUDE can be omitted, if it coincides with the altitude of the start point of the next segment

key = INTERMEDIATE_POINTS, value = value of type 'a(qddd)', that expresses an array of intermediate points

Note: an intermediate point is expressed as a struct(type,latitude,longitude,altitude), where type = enum(INVALID,HARD_POINT,SOFT_POINT, ...)

key = ROAD_NUMBER, value = value of type 's', that expresses the road number

key = ROAD_NAME, value = value of type 's', that expresses the road name

key = DISTANCE, value = value of type 'u', that identifies distance to the next segment in meters

key = TIME, value = value of type 'u', that identifies time to travel to the next segment in seconds

key = MANEUVER, value = value of type 'a(qqq)', that identifies a pictogram that describes the next maneuver (OPTIONAL)

key = INSTRUCTION, value = value of type 's', that identifies the instruction to the user

key = BORDER_CROSSING, value = value of type 's', that contains information about border crossings

key = TIME_ZONE, value = value of type 'n', that indicates the time zone of the current segment. It is expressed as the time difference from the UTC in minutes

key = DAYLIGHT_SAVING_TIME, value = value of type 'n', that indicates the daylight saving time of the current segment. It is expressed as the time difference from the UTC in minutes

key = ADDITIONAL_INFORMATION, value = value of type 's', that contains additional information to the user (toll cost, ...)

key = HIGHWAY_EXIT, value = value of type 's', that in case the road segment ends with a highway exit, it expresses the highway exit number

key = START_OFFSET, value = value of type 'u', that indicates the offset of the starting point in meters from the beginning of the route

Out aa{qv} routeSegments

GetRouteOverview = This method retrieves general information about a given route

method GetRouteOverview

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

valuesToReturn = array[key]

key =

enum(ARRIVAL_TIME,ARRIVAL_DATE,DEPARTURE_TIME,DEPARTURE_DATE,TOTAL_DISTANCE,TOTAL_TIME,
... ,ALL)

in aq valuesToReturn

routeOverview = array[detail]

detail = tuple(key,value)

key =

enum(ARRIVAL_TIME,ARRIVAL_DATE,DEPARTURE_TIME,DEPARTURE_DATE,TOTAL_DISTANCE,TOTAL_TIME,
...)

key = ARRIVAL_TIME, value of type 'u', that represents the arrival time expressed in seconds since mid-night (UTC)

key = ARRIVAL_DATE, value of type 'u', that represents the arrival date expressed either as calendar date (the number of days since 1 Jan 2000) or as weekday. The weekday is expressed with values from 0 to 6 (0 = Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday)

key = DEPARTURE_TIME, value of type 'u', that represents the departure time expressed in seconds since mid-night (UTC)

key = DEPARTURE_DATE, value of type 'u', that represents the departure date expressed either as calendar date (the number of days since 1 Jan 2000) or as weekday. The weekday is expressed with values from 0 to 6 (0 =

Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday)

key = TOTAL_DISTANCE, value of type 'u', that represents the total distance in m

key = TOTAL_TIME, value of type 'u', that represents the total time in seconds

out a{qv} routeOverview

GetRouteBoundingBox = This method retrieves the bounding box containing a calculated route

method GetRouteBoundingBox

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

boundingBox = struct(top-left-corner,bottom-right-corner)

geocoordinates of the top-left-corner = struct(lat,lon)

geocoordinates of the bottom-right-corner = struct(lat,lon)

lat = latitude in format %3.6f. Range[-90:+90]. Example: 48.053250

lon = longitude in format %3.6f. Range[-180:+180]. Example: 8.321000

out ((dd)(dd)) boundingBox

GetAllRoutes = This method retrieves the handles of all created routes

method GetAllRoutes

routesList = array[route]

route = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

out au routesList

AlternativeRoutesAvailable = This signal is emitted when alternative routes have been computed in the background and are available for guidance.

signal AlternativeRoutesAvailable

routeHandlesList = array[routeHandle]

routeHandle = Handle identifying a computed alternative route. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

out au routeHandlesList

SetBlockedRouteStretches = This method sets blocked stretches on a given route

method SetBlockedRouteStretches

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

routeHandle = Route handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

blockParameters = struct(offset,length)

offset = the offset in meters from the beginning of the route where the road block starts from

length = the length of the road block in meters

Note: pass an empty array to remove previously set blocked route stretches

in a(uu) blockParameters

GetBlockedRouteStretches = This method retrieves all blocked stretches on a given route

method GetBlockedRouteStretches

routeHandle = Route handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u routeHandle

blockParameters = struct(offset,length)

offset = the offset in meters from the beginning of the route where the road block starts from

length = the length of the road block in meters

Out a(uu) blockParameters

interface

org.genivi.navigationcore.Session

version 3.0.0 (22-01-2014)

Session = This interface offers functions to create and delete sessions

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

CreateSession = This method creates a new session

method CreateSession

client = name or identifier of the client application that requests a new session

The navigation core must internally associate this name to the returned session handle

This parameter can be used to identify the client application and determine if a given feature is enabled for it

in s client

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

out u sessionHandle

This error is generated if no more session handles are available

error org.genivi.navigationcore.Session.Error.NoMoreSessionHandles

DeleteSession = This method deletes a session and its associated resources

method DeleteSession

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

This error is generated if an application tries to delete a session handle that is not available

error org.genivi.navigationcore.Session.Error.SessionNotAvailable

GetSessionStatus = This method returns whether a given session handle is available or not (for example because it was deleted)

method GetSessionStatus

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle

sessionStatus = enum(INVALID,AVAILABLE,NOT_AVAILABLE)
out q sessionStatus

GetAllSessions = This method returns a list of all available sessions

method GetAllSessions

sessionsList = array[struct(sessionHandle,client)]
sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value
client = name or identifier of the client application that requested the sessionHandle
out a(us) sessionsList

SessionDeleted = This signal is emitted when a session is deleted

signal SessionDeleted

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value
out u sessionHandle

interface

org.genivi.navigationcore.Configuration

version 3.0.0 (21-01-2014)

Configuration = This interface offers functions to set and retrieve configuration parameters

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

SetUnitsOfMeasurement = This method sets the units of measurement

method SetUnitsOfMeasurement

unitsOfMeasurementList = array[unitsOfMeasurement]

unitsOfMeasurement = tuple (key,value)

key = enum(INVALID,LENGTH, ...)

key = LENGTH, value = value of type 'q', that represents an enum(INVALID,METER,MILE, ...)

in a{qv} unitsOfMeasurementList

GetUnitsOfMeasurement = This method retrieves the units of measurement

method GetUnitsOfMeasurement

unitsOfMeasurementList = array[unitsOfMeasurement]

unitsOfMeasurement = tuple (key,value)

key = enum(INVALID,LENGTH, ...)

key = LENGTH, value = value of type 'q', that represents an enum(INVALID,METER,MILE, ...)

out a{qv} unitsOfMeasurementList

GetSupportedUnitsOfMeasurement = This method retrieves the supported units of measurement

method GetSupportedUnitsOfMeasurement

unitsOfMeasurementList = array[unitsOfMeasurement]

unitsOfMeasurement = dictionary[key,value]

dictionary = array of tuples (key,value)

key = enum(INVALID,LENGTH, ...)

key = LENGTH, value = value of type 'aq'; 'q' is an enum(INVALID,METER,MILE, ...)

out a{qv} unitsOfMeasurementList

SetTimeFormat = This method sets the time format

method SetTimeFormat

timeFormat = enum(INVALID,12H,24H, ...)

in q timeFormat

GetTimeFormat = This method retrieves the time format

method GetTimeFormat

timeFormat = enum(INVALID,12H,24H, ...)

out q timeFormat

GetSupportedTimeFormats = This method retrieves the supported time formats

method GetSupportedTimeFormats

timeFormatList = array[timeFormat]

timeFormat = enum(INVALID,12H,24H, ...)

out aq timeFormatList

SetCoordinatesFormat = This method sets the coordinates format

method SetCoordinatesFormat

coordinatesFormat = enum(INVALID,DEGREES,MINUTES,SECONDS, ...)

DEGREES format = d.d°

MINUTES format = d°m.m'

SECONDS format = d°m's"

in q coordinatesFormat

GetCoordinatesFormat = This method retrieves the coordinates format

method GetCoordinatesFormat

coordinatesFormat = enum(INVALID,DEGREES,MINUTES,SECONDS, ...)

DEGREES format = d.d°

MINUTES format = d°m.m'

SECONDS format = d°m's"

out q coordinatesFormat

GetSupportedCoordinatesFormats = This method retrieves the supported coordinates formats

method GetSupportedCoordinatesFormats

coordinatesFormatList = array[coordinatesFormat]

coordinatesFormat = enum(INVALID,DEGREES,MINUTES,SECONDS, ...)
DEGREES format = d.d°
MINUTES format = d°m.m'
SECONDS format = d°m's"
out aq coordinatesFormatList

SetLocale = This method sets the current language and country

method SetLocale

languageCode = ISO 639-3 language code (lower case)
in s languageCode

countryCode = ISO 3166-1 alpha 3 country code (upper case)
in s countryCode

GetLocale = This method retrieves the current language and country

method GetLocale

languageCode = ISO 639-3 language code (lower case)
out s languageCode

countryCode = ISO 3166-1 alpha 3 country code (upper case)
out s countryCode

GetSupportedLocales = This method retrieves the supported languages and countries

method GetSupportedLocales

localeList = array[struct(languageCode,countryCode)]
languageCode = ISO 639-3 language code (lower case)
countryCode = ISO 3166-1 alpha 3 country code (upper case)
out a(ss) localeList

ConfigurationChanged = This signal is sent to the clients when one or more configuration settings changes

signal ConfigurationChanged

changedSettings = array[setting]
setting = enum(INVALID,UNITS_OF_MEASUREMENT,LOCALE,TIME_FORMAT,COORDINATES_FORMAT, ...)
out aq changedSettings

interface

org.genivi.navigationcore.Guidance

version 3.0.0 (21-01-2014)

Guidance = This interface offers functions that implement the route-guidance functionality of a navigation system

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

StartGuidance = This method starts the guidance for a given route
The guidanceStatus will change to ACTIVE
method StartGuidance

sessionHandle = Session handle. Range(0x0:0x7fffff). 0x0 is reserved as an invalid handle value
in u sessionHandle
routeHandle = Route handle. Range(0x0:0x7fffff). 0x0 is reserved as an invalid handle value
in u routeHandle

StopGuidance = This method stops the guidance
The guidanceStatus will change to INACTIVE
method StopGuidance

sessionHandle = Session handle. Range(0x0:0x7fffff). 0x0 is reserved as an invalid handle value
in u sessionHandle

SetVoiceGuidance = This method switch on/off the voice guidance
method SetVoiceGuidance

activation of the voice guidance
in b activate
kind of voice (to be defined)
in s voice
This error is generated if the voice generation is inactive
error org.genivi.navigationcore.Guidance.Error.VoiceNotAllowed

GetGuidanceDetails = This method retrieves guidance information
method GetGuidanceDetails

voiceGuidance = # TRUE voice guidance is active
out b voiceGuidance
vehicleOnTheRoad = # TRUE the vehicle is located on the road network
out b vehicleOnTheRoad
isDestinationReached = # TRUE the destination has been reached
out b isDestinationReached
maneuver = enum(INVALID,CRUISE,MANEUVER_APPEARED,PRE_ADVICE,ADVICE_PASSED,...)
out q maneuver

PlayVoiceManeuver = This method plays or repeats the last voice guidance
method PlayVoiceManeuver

This error is generated if the voice generation is inactive
error org.genivi.navigationcore.Guidance.Error.VoiceNotAllowed

GetWaypointInformation = This method retrieves the information on the remaining way points of the route. A point can be the final destination as well as a stage defined by the user. The returned waypoints are ordered by their 'number'.
method GetWaypointInformation

requestedNumberOfWaypoints = the number of requested waypoints. If 0, all waypoints will be returned.
in q requestedNumberOfWaypoints
numberOfWaypoints = the number of retrieved waypoints(NOTE: the number corresponds to the number of elements in the array)
out q numberOfWaypoints
array(struct(waypointOffset,travelTime,direction,side,timeZone,daylightSavingTime,isDestination,number])
waypointOffset = the offset of the way point in meters from the beginning of the route
travelTime = time to reach the way point in seconds
direction = direction of the way point in degree relatively to the North. Range [0;360]
side = enum(LEFT,RIGHT,NOT_AVAILABLE)
timeZone = time zone of the way point. It is expressed as the time difference from the UTC in minutes
daylightSavingTime = daylight saving time of the way point. It is expressed as the time difference from the UTC in minutes
isDestination = # TRUE the way point is the destination
number = number of the next waypoint (related to the waypoint list, first way point index is 0)
out a(uiuqnnbq) waypointsList

This method retrieves the information on the final destination
method GetDestinationInformation

offset = offset of the destination in meter from the beginning of the route
out u offset
travelTime = time to reach the destination in second
out u travelTime
direction = direction of the destination in degree relatively to the North. Range [0;360]
out i direction
side = enum(LEFT,RIGHT,NOT_AVAILABLE)
out q side
timeZone = time zone of the destination. It is expressed as the time difference from the UTC in minutes
out n timeZone
daylightSavingTime = daylight saving time of the destination. It is expressed as the time difference from the UTC in minutes
out n daylightSavingTime

VehicleLeftTheRoadNetwork = This signal is emitted when the vehicle exits from the road network
signal VehicleLeftTheRoadNetwork

GuidanceStatusChanged = This signal is emitted when the guidance status changes

signal GuidanceStatusChanged

guidanceStatus = enum(INVALID,ACTIVE,INACTIVE)

ACTIVE means that NavigationCore is providing guidance information

INACTIVE means that NavigationCore is not providing guidance information

out q guidanceStatus

routeHandle = Active route handle. Range(0x0:0x7fffff). 0x0 is reserved as an invalid handle value. Should be ignored when guidanceStatus=INACTIVE.

out u routeHandle

WaypointReached = This signal is emitted when the destination is reached

signal WaypointReached

isDestination = flag. TRUE means that the way point is the destination

out b isDestination

This signal is emitted each time a maneuver event is going

signal ManeuverChanged

maneuver = enum(INVALID,CRUISE,MANEUVER_APEARED_PRE_ADVICE,ADVICE_PASSED,...)

out q maneuver

PositionOnRouteChanged = This signal is emitted when the position on the route changes

signal PositionOnRouteChanged

offsetOnRoute = the current offset on the route in meters from the beginning of the route

out u offsetOnRoute

GetManeuversList = This method retrieves the list of next maneuvers

method GetManeuversList

requestedNumberOfManeuvers = the number of requested maneuvers

in q requestedNumberOfManeuvers

maneuverOffset = the offset of the first maneuver to retrieve

in u maneuverOffset

numberOfManeuvers = the number of retrieved maneuvers

Note: the number corresponds to the number of elements in the array

out q numberOfManeuvers

maneuversList =

array(struct(roadNumberAfterManeuver,roadNameAfterManeuver,roadPropertyAfterManeuver,drivingSide,offsetOfNextManeuver,maneuverDetails))

roadNumberAfterManeuver = the number of the road after the maneuver (if a road has multiple road numbers,

they will be separated by slashes (/) and combined into one string)

roadNameAfterManeuver = the name of the road after the maneuver

roadPropertyAfterManeuver = enum(TOLL_ROADS,...,DEFAULT)

drivingSide = enum(LEFT,RIGHT)

offsetOfNextManeuver = the offset of the next maneuver in meters from the beginning of the route (next maneuver

is the second maneuver on the route ahead)

maneuverDetails = array(struct(offsetOfManeuver,travelTime,direction,maneuver,maneuverDirectionDetails))

offsetOfManeuver = the offset of the current maneuver in meters from the beginning of the route (current

maneuver is the first maneuver on the route ahead)

travelTime = travel time to the basic maneuver in seconds

direction = direction of the maneuver in degree relatively to the North. Range [0:360]

maneuver =

enum(INVALID,STRAIGHT_ON,TURN,CROSSROAD,ROUNDABOUT,HIGHWAY_ENTER,HIGHWAY_EXIT,BIFURCATION,FOLLOW_SPECIFIC_LANE,DESTINATION,WAYPOINT,ROAD_FORM_CHANGE)

maneuverDirectionDetails = array(struct(key,value))

key = enum(LENGTH,DIRECTION,EXIT_NUMBER,ROAD_FORM,LANE_INFO)

key = LENGTH, value of type 'q', when maneuver=ROUNDABOUT, expresses the length of the route segment

between the entry to and the exit from the roundabout

key = DIRECTION, value of type 'q',

enum(INVALID,STRAIGHT_ON,LEFT,SLIGHT_LEFT,HARD_LEFT,RIGHT,SLIGHT_RIGHT,HARD_RIGHT,UTURN_RIGHT,UTURN_LEFT)

key = EXIT_NUMBER, when maneuver=ROUNDABOUT, value of type 'q' that expresses the roundabout exit

number

when maneuver=HIGHWAY_EXIT, value of type 's' that expresses the highway exit number

key = ROAD_FORM, value of type 'q',

enum(INVALID,ROAD_REGULAR,ROAD_HIGHWAY,MOTORWAY,ROAD_FERRY)

key = LANE_INFO, value of type 's(uuuq)', array(struct(laneIndex,laneDirections,directionToFollow,divider))

laneIndex = number of the individual lane. Counting starts from zero, beginning at the left-most lane in the direction

of travel (independent of the driving side)

laneDirections = bitmask where each bit corresponds to a certain direction. A 1-bit indicates that the corresponding

part of the lane arrow is drawn in the lane information on the street (see the lane info bitmasks)

directionToFollow = bitmask where each bit corresponds to a certain direction. A 1-bit indicates that the

corresponding part of the lane arrow matches the direction of the corresponding maneuver (see the lane info

bitmasks). At most one bit of this bitmask will be set.

The bitmasks are:

LANE_INFO_BITMASK_STRAIGHT,LANE_INFO_BITMASK_SLIGHTRIGHT,LANE_INFO_BITMASK_RIGHT,LANE_INFO_BITMASK_SHARPRIGHT,LANE_INFO_BITMASK_RIGHTUTURN,LANE_INFO_BITMASK_SLIGHTLEFT,LANE_INFO_BITMASK_LEFT,LANE_INFO_BITMASK_SHARPLEFT,LANE_INFO_BITMASK_LEFTUTURN

divider = indicates the type of divider between the given lane and the next one to the right, in the direction of travel

enum(DIVIDER_UNDEFINED,DIVIDER_INTERRUPTEDLONG,DIVIDER_INTERRUPTEDSHORT,DIVIDER_SOLID,SINGLE,DIVIDER_SOLIDDOUBLE,DIVIDER_SOLIDINTERRUPTED,DIVIDER_INTERRUPTEDSOLID)

Note: To describe the divider on the left side of the left-most lane, use the following entry in LANE_INFO:

(laneIndex=0x7fffff, laneDirections=0x00000000, directionToFollow=0x00000000, divider=type)

out a(sssqqaa(uiuiqa(qv))) maneuversList

This error is generated in case there's no maneuver until the destination

error org.genivi.navigationcore.Guidance.Error.NoManeuver

VehicleLeftTheRoute = This signal is emitted when the vehicle has left the route

signal VehicleLeftTheRoute

SetRouteCalculationMode = This method configures the way the navigation application wants the navigation core to behave

of reroute trigger

method SetRouteCalculationMode

sessionHandle = Session handle. Range(0x0:0x7fffff). 0x0 is reserved as an invalid handle value

in u sessionHandle

routeCalculationMode =

enum(INVALID,ALL_MANUAL,ALL_AUTOMATIC,TRAFFIC_MANUAL,OFF_ROUTE_MANUAL)

in q routeCalculationMode

SkipNextManeuver = This method allows to jump behind the current maneuver

method SkipNextManeuver

sessionHandle = Session handle. Range(0x0:0x7fffff). 0x0 is reserved as an invalid handle value

in u sessionHandle

This error is generated in case there's no maneuver until the destination

error org.genivi.navigationcore.Guidance.Error.NoManeuver

GetGuidanceStatus = This method retrieves the guidance status

method GetGuidanceStatus

guidanceStatus = enum(INVALID,ACTIVE,INACTIVE)

ACTIVE means that NavigationCore is providing guidance information

INACTIVE means that NavigationCore is not providing guidance information

out q guidanceStatus

routeHandle = Active route handle. Range(0x0:0x7fffff). 0x0 is reserved as an invalid handle value. Should be

ignored when guidanceStatus=INACTIVE
out u routeHandle

SetVoiceGuidanceSettings = This method sets the voice guidance settings
method SetVoiceGuidanceSettings

mode = enum(INVALID,DISABLED_PROMPT,AUTOMATIC_PROMPT,MANUAL_PROMPT,...)
MANUAL_PROMPT means that a client application can ask the NavigationCore to play the voice prompts
AUTOMATIC_PROMPT means that the voice prompts will be requested by NavigationCore automatically
DISABLED_PROMPT means that the client application will the voice generator component directly to play the messages (bypassing the NavigationCore)
in q promptMode

GetVoiceGuidanceSettings = This method returns the used voice guidance settings
method GetVoiceGuidanceSettings

mode = enum(INVALID,DISABLED_PROMPT,AUTOMATIC_PROMPT,MANUAL_PROMPT,...)
MANUAL_PROMPT means that a client application can ask the NavigationCore to play the voice prompts
AUTOMATIC_PROMPT means that the voice prompts will be requested by NavigationCore automatically
DISABLED_PROMPT means that the client application will the voice generator component directly to play the messages (bypassing the NavigationCore)
out q promptMode

PositionToRouteChanged = This signal is emitted when the vehicle is off-the-road network and either the heading or the distance (or both) to the closest point on the active route changes
signal PositionToRouteChanged

distance = distance in meters to the closest point on the active route
out u distance

direction = direction in degrees relatively to the closest point on the active route. Range [0;360]
out i direction

ActiveRouteChanged = This signal is emitted when the active route changes
signal ActiveRouteChanged

changeCause = enum(INVALID,TRAFFIC,OFF_ROUTE,MANUAL,...)
out q changeCause

interface

org.genivi.navigationcore.MapMatchedPosition

version 3.0.0 (21-01-2014)

MapMatchedPosition = This interface offers functions to retrieve the map matched position and to simulate positioning

If NavigationCore is not in Simulation Mode (Simulation Status is SIMULATION_STATUS_NO_SIMULATION), it is using the EnhancedPosition from the Positioning component.

In Simulation Mode it is not using this position, instead it uses FixedPosition or FollowActiveRoute to determine the position. With FixedPosition (Simulation Status is SIMULATION_STATUS_FIXED_POSITION), the position is fixed, unless it is changed by a call to setPosition().

This supports use cases like: setting the current car position in a demo mode, or replay a position log file (where setPosition() is called for each logged location).

In Follow Active Route mode, NavigationCore is generating positions itself.

These positions follow the current active route. When the end of the route is reached, the position jumps back to the starting point of the route.

There are two sub states: Running (Simulation Status is SIMULATION_STATUS_RUNNING) and Paused (Simulation Status is SIMULATION_STATUS_PAUSED).

By default the 'driving speed' will be equal to the free flow speed of each road segment. However a speed factor can be set via the method SetSimulationSpeed.

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

SetSimulationMode = This method activates or deactivates the simulation mode

method SetSimulationMode

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

activate = flag. TRUE means that the simulation mode is activated.

The simulation status will be SIMULATION_STATUS_FIXED_POSITION, with the position being the last known position in the NavigationCore.

FALSE means that the simulation mode is de-activated. The simulation status will be SIMULATION_STATUS_NO_SIMULATION

in b activate

GetSimulationStatus = This method retrieves the simulation status

method GetSimulationStatus

simulationStatus = enum(SIMULATION_STATUS_NO_SIMULATION, SIMULATION_STATUS_RUNNING, SIMULATION_STATUS_PAUSED, SIMULATION_STATUS_FIXED_POSITION)

SIMULATION_STATUS_NO_SIMULATION means that NavigationCore is using the EnhancedPosition

SIMULATION_STATUS_RUNNING means that positions are generated along the active route

SIMULATION_STATUS_PAUSED means that the generation of positions along the active route is paused

SIMULATION_STATUS_FIXED_POSITION means that the position is fixed.

out q simulationStatus

AddSimulationStatusListener = Add this node as a listener to Simulation Status changes.

Upon changes a SimulationStatusChanged signal will be received. NavigationCore will only send out a SimulationStatusChanged signal if there is at least one node listening to these changes.

method AddSimulationStatusListener

RemoveSimulationStatusListener = Remove this node as a listener to Simulation Status changes.

method RemoveSimulationStatusListener

SimulationStatusChanged = This signal is emitted when the Simulation Status has changed

signal SimulationStatusChanged

simulationStatus = enum(SIMULATION_STATUS_NO_SIMULATION, SIMULATION_STATUS_RUNNING, SIMULATION_STATUS_PAUSED, SIMULATION_STATUS_FIXED_POSITION)

SIMULATION_STATUS_NO_SIMULATION means that NavigationCore is using the EnhancedPosition

SIMULATION_STATUS_RUNNING means that positions are generated along the active route

SIMULATION_STATUS_PAUSED means that the generation of positions along the active route is paused

SIMULATION_STATUS_FIXED_POSITION means that the position is fixed.

out q simulationStatus

SetSimulationSpeed = This method sets the speed factor for the simulation mode

method SetSimulationSpeed

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

speedFactor = speed factor

unit is x0.25. Normal speed x1 is 4

in y speedFactor

GetSimulationSpeed = returns the speed factor for the simulation mode

method GetSimulationSpeed

speedFactor = speed factor

unit is x0.25. Normal speed x1 is 4

out y speedFactor

AddSimulationSpeedListener = Add this node as a listener to simulation speed factor changes.

Upon changes a SimulationSpeedChanged signal will be received.

NavigationCore will only send out a SimulationSpeedChanged signal if there is at least one node listening to these changes

method AddSimulationSpeedListener

RemoveSimulationSpeedListener = Remove this node as a listener to simulation speed factor changes.

method RemoveSimulationSpeedListener

SimulationSpeedChanged = This signal is emitted when the simulation speed factor has changed

NavigationCore will only send out a SimulationSpeedChanged signal if there is at least one node listening to these changes

signal SimulationSpeedChanged

speedFactor = speed factor

unit is x0.25. Normal speed x1 is 4

out y speedFactor

StartSimulation = This method starts, or resumes, a Follow Active Route simulation

If the current Simulation Status is SIMULATION_STATUS_PAUSED, the simulation is resumed from the current location.

Otherwise the simulation is started from the starting point of the route. In both cases the new status will be

SIMULATION_STATUS_RUNNING

method StartSimulation

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

PauseSimulation = This method freezes the current location

The new status will be SIMULATION_STATUS_PAUSED

method PauseSimulation

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

GetPosition = This method returns the current position

method GetPosition

valuesToReturn= array[key]

key =

enum(INVALID,TIMESTAMP,LATITUDE,LONGITUDE,ALTITUDE,HEADING,SPEED,CLIMB,GNSS_FIX_STATUS,DR_STATUS,MM_STATUS,SIMULATION_MODE,

... ,ALL)

in aq valuesToReturn

position = dictionary[key,value]

dictionary = array of tuples (key,value)

key =

enum(INVALID,TIMESTAMP,LATITUDE,LONGITUDE,ALTITUDE,HEADING,SPEED,CLIMB,GNSS_FIX_STATUS,DR_STATUS,MM_STATUS,SIMULATION_MODE,

...)

key = TIMESTAMP, value = value of type 't', that represents a timestamp in ms

key = LATITUDE, value = value of type 'd', that expresses the latitude of the current position in format %3.6f.

Range [-90,+90]. Example: 48.053250

key = LONGITUDE, value = value of type 'd', that expresses the longitude of the current position in format %3.6f.

Range [-180,+180]. Example: 8.324500

key = ALTITUDE, value = value of type 'i', that expresses the altitude above the sea level of the current position in meters

key = HEADING, value = value of type 'u', that expresses the course angle in degree (0 = north, 90 = east, 180 = south, 270 = west, no negative values)
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 'f', that expresses the inclination measured in degrees
key = GNSS_FIX_STATUS, value = value of type 'q', that represents an
enum(INVALID,NO_FIX,TIME_FIX,2D_FIX,3D_FIX, ...)
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 = MM_STATUS, value = value of type 'b', where TRUE means that a map-matching algorithm has been used to calculate the current position
key = SIMULATION_MODE, value = value of type 'b', where TRUE means that the current position is simulated
Out a{qv} position

This error is generated if no position is available

error org.genivi.navigationcore.MapMatchedPosition.Error.NoPosition

SetPosition = This method sets the position to a specific location

Independent of the current Simulation Status, the new status will be SIMULATION_STATUS_FIXED_POSITION.

This method can be used to replay a position log file (with positions obtained via calls to GetPosition()) by calling this method for each position in the log file.

It is of course also possible to call this method just once with e.g. a 'current location' entered by the user (via the HMI).

method SetPosition

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

position = dictionary[key,value]

dictionary = array of tuples (key,value)

key =

enum(INVALID,TIMESTAMP,LATITUDE,LONGITUDE,ALTITUDE,HEADING,SPEED,CLIMB,GNSS_FIX_STATUS,DR_STATUS,MM_STATUS, ...)

key = TIMESTAMP, value = value of type 't', that represents a timestamp in ms

key = LATITUDE, value = value of type 'd', that expresses the latitude of the current position in format %3.6f.

Range [-90,+90]. Example: 48.053250

key = LONGITUDE, value = value of type 'd', that expresses the longitude of the current position in format %3.6f.

Range [-180,+180]. Example: 8.324500

key = ALTITUDE, value = value of type 'f', that expresses the altitude above the sea level of the current position in meters

key = HEADING, value = value of type 'u', that expresses the course angle in degree (0 = north, 90 = east, 180 = south, 270 = west, no negative values)

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 'f', that expresses the inclination measured in degrees

key = GNSS_FIX_STATUS, value = value of type 'q', that represents an

enum(INVALID,NO_FIX,TIME_FIX,2D_FIX,3D_FIX, ...)

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 = MM_STATUS, value = value of type 'b', where TRUE means that a map-matching algorithm has been used to calculate the current position

Note that the key SIMULATION_MODE is not allowed here, as it will be true by definition.

in a{qv} position

PositionUpdate = This signal is called to notify a client application of a position change. The update frequency is implementation specific. The maximal allowed frequency is 10Hz

signal PositionUpdate

changedValues = array[value]

value =

enum(INVALID,TIMESTAMP,LATITUDE,LONGITUDE,ALTITUDE,HEADING,SPEED,CLIMB,GNSS_FIX_STATUS,DR_STATUS,MM_STATUS,SIMULATION_MODE, ...)

Out aq changedValues

GetAddress = This method returns the current address

method GetAddress

valuesToReturn= array[fieldType]

key =

enum(INVALID,TIMESTAMP,COUNTRY,COUNTRYCODE,CITY,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,TIMEZONE_OFFSET,DAYLIGHT_OFFSET,MATCH_TYPE, ... ,ALL)

in aq valuesToReturn

address = dictionary[key,value]

dictionary = array of tuples (key,value)

key =

enum(INVALID,TIMESTAMP,COUNTRY,COUNTRYCODE,CITY,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,TIMEZONE_OFFSET,DAYLIGHT_OFFSET,MATCH_TYPE, ...)

key = TIMESTAMP, value = value of type 't', that represents a timestamp in ms

key = COUNTRY, value = value of type 's', that identifies the country name

key = COUNTRYCODE, value = value of type 's', ISO 3166-1 alpha 3 country code (upper case)

key = CITY, value = value of type 's', that identifies the city name

key = STREET, value = value of type 's', that identifies the street name

key = ROAD_NUMBER, value = value of type 's', that identifies the road number

key = HOUSENUMBER, value = value of type 's', that identifies the house number

key = HOUSENAME, value = value of type 's', that identifies the house name
key = CROSSING, value = value of type 's', that identifies the crossing
key = DISTRICT, value = value of type 's', that identifies the district name
key = TIMEZONE_OFFSET, value = value of type 'n', that identifies the timezone offset at the current address
key = DAYLIGHT_OFFSET, value = value of type 'n', that identifies the daylight offset at the current address
key = MATCH_TYPE, value = value of type 'q', that identifies an
enum(INVALID,ON_ROAD,OFF_ROAD,ON_FERRY,IN_TUNNEL,ON_CARPARK, ...)
Out a{qv} address

This error is generated if no map is available

error org.genivi.navigationcore.MapMatchedPosition.Error.NoMap

This error is generated if the vehicle is located in a position outside of the known map

error org.genivi.navigationcore.MapMatchedPosition.Error.OutOfKnownMap

AddressUpdate = This signal is called to notify a client application that the current address changed

signal AddressUpdate

changedValues = array[value]
value =
enum(INVALID,TIMESTAMP,COUNTRY,COUNTRYCODE,CITY,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,TIMEZONE_OFFSET,DAYLIGHT_OFFSET,MATCH_TYPE,
...)
Out aq changedValues

positionOnSegment = This method returns the vehicle position on a route segment

method GetPositionOnSegment

valuesToReturn = array[fieldType]
key = enum(INVALID,TIMESTAMP,SEGMENT_ID,DIRECTION_ON_SEGMENT,DISTANCE_ON_SEGMENT, ...
,ALL)
in aq valuesToReturn

positionOnSegment = dictionary[key,value]
dictionary = array of tuples (key,value)
key = enum(INVALID,TIMESTAMP,SEGMENT_ID,DIRECTION_ON_SEGMENT,DISTANCE_ON_SEGMENT, ...)
key = TIMESTAMP, value = value of type 't', that represents a timestamp in ms
key = SEGMENT_ID, value = value of type 'ay', that represents a link-ID in a format whose interpretation is left to the navigationcore
key = DIRECTION_ON_SEGMENT, value = value of type 'b', where TRUE means forward and FALSE means backward
key = DISTANCE_ON_SEGMENT, value = value of type 'd', that indicates the distance in meter on the segment
Out a{qv} positionOnSegment

PositionOnSegmentUpdate = This signal is called to notify the client that the vehicle position on the route segment changed

signal PositionOnSegmentUpdate

changedValues = array[value]
value = enum(INVALID,TIMESTAMP,SEGMENT_ID,DIRECTION_ON_SEGMENT,DISTANCE_ON_SEGMENT, ...)
Out aq changedValues

GetStatus = This method returns the current status

method GetStatus

valuesToReturn = array[fieldType]
key = enum(INVALID,TIMESTAMP,GNSS_FIX_STATUS,DR_STATUS,MM_STATUS,SIMULATION_MODE, ... ,ALL)
in aq valuesToReturn

status = dictionary[key,value]
dictionary = array of tuples (key,value)
key = enum(INVALID,TIMESTAMP,GNSS_FIX_STATUS,DR_STATUS,MM_STATUS,SIMULATION_MODE, ...)
key = TIMESTAMP, value = value of type 't', that represents a timestamp in ms
key = GNSS_FIX_STATUS, value = value of type 'q', that represents an
enum(INVALID,NO_FIX,TIME_FIX,2D_FIX,3D_FIX, ...)
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 = MM_STATUS, value = value of type 'b', where TRUE means that a map-matching algorithm has been used to calculate the current position
key = SIMULATION_MODE, value = value of type 'b', where TRUE means that the current position is simulated
Out a{qv} status

StatusUpdate = This signal is emitted to notify a client application that the current status changed

signal StatusUpdate

changedValues = array[value]
value = enum(INVALID,TIMESTAMP,GNSS_FIX_STATUS,DR_STATUS,MM_STATUS,SIMULATION_MODE, ...)
Out aq changedValues

OffroadPositionChanged = This signal is emitted when the heading and the distance to the closest point on the road network changes

signal OffRoadPositionChanged

distance = distance in meters to the closest point on the road network

out u distance

direction = direction in degrees relatively to the closest point on the road network. Range [0:360]

out i direction

interface

org.genivi.navigationcore.LocationInput

version 3.0.0 (21-01-2014)

LocationInput = This interface offers functions that implement the location-input functionality of a navigation system

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

method GetVersion

```
version = struct(major,minor,micro,date)
major = when the major version changes, then backward compatibility with previous releases is not granted
minor = when the minor version 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 version 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
```

CreateLocationInput = This method creates a new location input and retrieves a handle

method CreateLocationInput

```
sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle
```

```
locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
out u locationInputHandle
```

This error is generated if no more location input handles are available

error org.genivi.navigationcore.LocationInput.Error.NoMoreLocationInputHandles

DeleteLocationInput = This method deletes a location input and its associated resources

method DeleteLocationInput

```
sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle
```

```
locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u locationInputHandle
```

This error is generated if an application tries to delete a location input handle that is not available

error org.genivi.navigationcore.LocationInput.Error.LocationInputNotAvailable

GetSupportedAddressAttributes = This method retrieves the supported address attributes

method GetSupportedAddressAttributes

```
addressAttributesList = array[attribute]
attribute =
enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,STATE,CITY,ZIPCODE,STREET,HOUSENUMBER,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)
out aq addressAttributesList
```

SetAddress = This method sets the address to start with for the LocationInput identified by the given handle

method SetAddress

```
sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle
```

```
locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u locationInputHandle
```

```
address = array[attribute]
attribute = tuple(key,value)
```

```
key =
```

```
enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)
```

```
key = LATITUDE, value = value of type 'd', that expresses the latitude in format %3.6f. Range[-90:+90]. Example: 48.70901
```

```
key = LONGITUDE, value = value of type 'd', that expresses the longitude in format %3.6f. Range[-180:+180]. Example: 9.167898
```

```
key = ALTITUDE, value = value of type 'i', that expresses the altitude in meters
```

```
key = COUNTRY, value = value of type 's', that identifies the country name
```

```
key = COUNTRYCODE, value = value of type 's', ISO 3166-1 alpha 3 country code (upper case)
```

```
key = CITY, value = value of type 's', that identifies the city name
```

```
key = STREET, value = value of type 's', that identifies the street name
```

```
key = ROAD_NUMBER, value = value of type 's', that identifies the road number
```

```
key = HOUSENUMBER, value = value of type 's', that identifies the house number
```

```
key = HOUSENAME, value = value of type 's', that identifies the house name
```

```
key = CROSSING, value = value of type 's', that identifies the crossing
```

```
key = DISTRICT, value = value of type 's', that identifies the district name
```

```
key = PHONENUMBER, value = value of type 's', that identifies a phone number
```

```
key = POINAME, value = value of type 's', that identifies a POI name
```

```
in a(qv) address
```

SetSelectionCriterion = This method sets the selection criterion for the current speller, search input and the corresponding result-lists for the current session

method SetSelectionCriterion

```
sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
in u sessionHandle
```

```
locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
```

in u locationInputHandle

selectionCriterion =

enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)

in q selectionCriterion

Spell = This method sends the next spell input for the current session

Note: when a spell is started the entries of the search are removed

method Spell

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u locationInputHandle

inputString = last input character (UTF-8) (0x08(Backspace) for delete last character, 0x0D(Carriage Return) for delete entire input)

in s inputCharacter

maxWindowSize = maximum number of elements that should be returned as result

in q maxWindowSize

Search = This method sends the search input for the current session

Note: when a search is started the entries of the spell input are removed

method Search

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u locationInputHandle

inputString = contains the String, that is searched

in s inputString

maxWindowSize = maximum number of elements that should be returned as result

in q maxWindowSize

CurrentSelectionCriterion = This signal notifies the SelectionCriterion for the current speller input or search.

Note: when no SelectionCriterion was set or an input was finished, the SelectionCriterion has the value INVALID

signal CurrentSelectionCriterion

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

out u locationInputHandle

selectionCriterion =

enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)

out q selectionCriterion

SearchStatus = This signal updates the search status of the specified session

signal SearchStatus

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

out u locationInputHandle

statusValue = enum(INVALID,NOT_STARTED,SEARCHING,FINISHED, ...)

out q statusValue

SpellResult = This signal notifies the result of the previous Spell method

signal SpellResult

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

out u locationInputHandle

uniqueString = unique string derived from spell input (i.e. including auto-completion if applicable)

out s uniqueString

validCharacters = set of (UTF-8 encoded) characters valid for next input (unified in a single string). A

Backspace(0x08) is returned if the input character passed to the Spell method was invalid

out s validCharacters

fullMatch = flag indicating whether the value in UniqueCharacters is already a full match for an existing list entry

out b fullMatch

RequestListUpdate = This method sends a request for more list elements for the current session

method RequestListUpdate

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u locationInputHandle

offset = starting offset of the newly requested list elements

in q offset

maxWindowSize = maximum number of elements that should be returned as result

in q maxWindowSize

SearchResultList = This signal updates the address result list (e.g. after a Search/Spell/Scroll call)

signal SearchResultList

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
Out u locationInputHandle

totalSize = total size of the result list
Out q totalSize

windowOffset = window offset within the complete list
Out q windowOffset

windowSize = size of the provided window
Out q windowSize

resultListWindow = array[address]
address = array[attribute]
attribute = tuple(key,value)
key =
enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)
key = LATITUDE, value = value of type 'd', that expresses the latitude in format %3.6f. Range [-90:+90]. Example:
48.70901
key = LONGITUDE, value = value of type 'd', that expresses the longitude in format %3.6f. Range [-180:+180].
Example: 9.167898
key = ALTITUDE, value = value of type 'i', that expresses the altitude in meters
key = COUNTRY, value = value of type 's', that identifies the country name
key = COUNTRYCODE, value = value of type 's', ISO 3166-1 alpha 3 country code (upper case)
key = CITY, value = value of type 's', that identifies the city name
key = STREET, value = value of type 's', that identifies the street name
key = ROAD_NUMBER, value = value of type 's', that identifies the road number
key = HOUSENUMBER, value = value of type 's', that identifies the house number
key = HOUSENAME, value = value of type 's', that identifies the house name
key = CROSSING, value = value of type 's', that identifies the crossing
key = DISTRICT, value = value of type 's', that identifies the district name
key = PHONENUMBER, value = value of type 's', that identifies a phone number
key = POINAME, value = value of type 's', that identifies a POI name
Out aa{qv} resultListWindow

SearchResultListSizeChanged = This signal updates the size of the address result list

signal SearchResultListSizeChanged

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
Out u locationInputHandle

totalSize = total size of the result list
Out q totalSize

SelectEntry = This method triggers selection of a result list entry by index

Note: the update of the input content will be notified in signal ContentUpdated

method SelectEntry

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
In u sessionHandle

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
In u locationInputHandle

index = absolute list index of the entry to be selected
In q index

GetEntry = This method synchronously gets the address for the given result list entry

method GetEntry

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
In u locationInputHandle

index = list index of the entry to be returned
In q index

address = array[attribute]
attribute = tuple(key,value)
key =
enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)
key = LATITUDE, value = value of type 'd', that expresses the latitude in format %3.6f. Range[-90:+90]. Example:
48.70901
key = LONGITUDE, value = value of type 'd', that expresses the longitude in format %3.6f. Range[-180:+180].
Example: 9.167898
key = ALTITUDE, value = value of type 'i', that expresses the altitude in meters
key = COUNTRY, value = value of type 's', that identifies the country name
key = COUNTRYCODE, value = value of type 's', ISO 3166-1 alpha 3 country code (upper case)
key = CITY, value = value of type 's', that identifies the city name
key = STREET, value = value of type 's', that identifies the street name
key = ROAD_NUMBER, value = value of type 's', that identifies the road number
key = HOUSENUMBER, value = value of type 's', that identifies the house number
key = HOUSENAME, value = value of type 's', that identifies the house name
key = CROSSING, value = value of type 's', that identifies the crossing
key = DISTRICT, value = value of type 's', that identifies the district name
key = PHONENUMBER, value = value of type 's', that identifies a phone number
key = POINAME, value = value of type 's', that identifies a POI name
Out a{qv} address

ContentUpdated = This signal updates the input content data for the specified session

signal ContentUpdated

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value
Out u locationInputHandle

guidable = flag indicating whether the current address is guidable

Out b guidable

availableSelectionCriteria = array of

enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)

Note: availableSelectionCriteria indicates the parts of the address that can be changed

Out aq availableSelectionCriteria

address = array[attribute]

attribute = tuple(key,value)

key =

enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)

key = LATITUDE, value = value of type 'd', that expresses the latitude in format %3.6f. Range[-90:+90]. Example:

48.70901

key = LONGITUDE, value = value of type 'd', that expresses the longitude in format %3.6f. Range[-180:+180].

Example: 9.167898

key = ALTITUDE, value = value of type 'i', that expresses the altitude in meters

key = COUNTRY, value = value of type 's', that identifies the country name

key = COUNTRYCODE, value = value of type 's', ISO 3166-1 alpha 3 country code (upper case)

key = CITY, value = value of type 's', that identifies the city name

key = STREET, value = value of type 's', that identifies the street name

key = ROAD_NUMBER, value = value of type 's', that identifies the road number

key = HOUSENUMBER, value = value of type 's', that identifies the house number

key = HOUSENAME, value = value of type 's', that identifies the house name

key = CROSSING, value = value of type 's', that identifies the crossing

key = DISTRICT, value = value of type 's', that identifies the district name

key = PHONENUMBER, value = value of type 's', that identifies a phone number

key = POINAME, value = value of type 's', that identifies a POI name

Out a(qv) address

ValidateAddress = This method validates an address from different sources than Navigation

method ValidateAddress

sessionHandle = Session handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

in u locationInputHandle

inputAddress = array[attribute]

attribute = tuple(key,value)

key =

enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)

key = LATITUDE, value = value of type 'd', that expresses the latitude in format %3.6f. Range[-90:+90]. Example:

48.70901

key = LONGITUDE, value = value of type 'd', that expresses the longitude in format %3.6f. Range[-180:+180].

Example: 9.167898

key = ALTITUDE, value = value of type 'i', that expresses the altitude in meters

key = COUNTRY, value = value of type 's', that identifies the country name

key = COUNTRYCODE, value = value of type 's', ISO 3166-1 alpha 3 country code (upper case)

key = CITY, value = value of type 's', that identifies the city name

key = STREET, value = value of type 's', that identifies the street name

key = ROAD_NUMBER, value = value of type 's', that identifies the road number

key = HOUSENUMBER, value = value of type 's', that identifies the house number

key = HOUSENAME, value = value of type 's', that identifies the house name

key = CROSSING, value = value of type 's', that identifies the crossing

key = DISTRICT, value = value of type 's', that identifies the district name

key = PHONENUMBER, value = value of type 's', that identifies a phone number

key = POINAME, value = value of type 's', that identifies a POI name

in a(qv) inputAddress

AddressValidationResult = This signal notifies the validation result of a former ValidateAddress call

signal AddressValidationResult

locationInputHandle = Location input handle. Range[0x0:0x7fffffff]. 0x0 is reserved as an invalid handle value

Out u locationInputHandle

validatedAddressList = array[validatedAddress]

validatedAddress = array[attribute]

attribute = tuple(key,value)

key =

enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)

key = LATITUDE, value = value of type 'd', that expresses the latitude in format %3.6f. Range[-90:+90]. Example:

48.70901

key = LONGITUDE, value = value of type 'd', that expresses the longitude in format %3.6f. Range[-180:+180].

Example: 9.167898

key = ALTITUDE, value = value of type 'i', that expresses the altitude in meters

key = COUNTRY, value = value of type 's', that identifies the country name

key = COUNTRYCODE, value = value of type 's', ISO 3166-1 alpha 3 country code (upper case)

key = CITY, value = value of type 's', that identifies the city name

key = STREET, value = value of type 's', that identifies the street name

key = ROAD_NUMBER, value = value of type 's', that identifies the road number

key = HOUSENUMBER, value = value of type 's', that identifies the house number

key = HOUSENAME, value = value of type 's', that identifies the house name

key = CROSSING, value = value of type 's', that identifies the crossing

key = DISTRICT, value = value of type 's', that identifies the district name

key = PHONENUMBER, value = value of type 's', that identifies a phone number

key = POINAME, value = value of type 's', that identifies a POI name

Out aa(qv) validatedAddressList

validationStatusList = array[validationStatus]

validationStatus = array[tem]

item = tuple(key,value)

key =

enum(INVALID,LATITUDE,LONGITUDE,ALTITUDE,FULL_ADDRESS,COUNTRY,COUNTRYCODE,STATE,CITY,ZIPCODE,STREET,ROAD_NUMBER,HOUSENUMBER,HOUSENAME,CROSSING,DISTRICT,PHONENUMBER,POINAME,TOWNCENTER,
...)

value = enum(INVALID,OK,UNKNOWN,AMBIGUOUS,INCONSISTENT)

Out aa(qq) validationStatusList

ReverseGeocode = This method transforms a geocoordinate into an address

Note: the update of the input content will be notified in signal *ContentUpdated*

method ReverseGeocode

sessionHandle = Session handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u sessionHandle

locationInputHandle = Location input handle. Range[0x0:0x7ffffff]. 0x0 is reserved as an invalid handle value

in u locationInputHandle

coordinate = struct{lat,lon}

lat = latitude in format %3.6f. Range[-90:+90]. Example: 48.053250

lon = longitude in format %3.6f. Range[-180:+180]. Example: 8.321000

in (dd) coordinate

constants *NavigationCore* version 3.0.0 (21-01-2014)

• *This document defines the constants used in the NavigationCore APIs*

• *INVALID = 0x0000*

• *DEFAULT = 0xffffe*

• *ALL = 0xffff*

• *AVAILABLE = 0x0001*

• *NOT_AVAILABLE = 0x0002*

• *TIME_FORMAT = 0x0003*

• *12H = 0x0004*

• *24H = 0x0005*

• *COORDINATES_FORMAT = 0x0006*

• *DEGREES = 0x0007*

• *MINUTES = 0x0008*

• *SECONDS = 0x0009*

• *TIMESTAMP = 0x0010*

• *TIMEZONE_OFFSET = 0x0011*

• *DAYLIGHT_OFFSET = 0x0012*

• *LOCALE = 0x0025*

• *UNITS_OF_MEASUREMENT = 0x0030*

• *LENGTH = 0x0031*

• *METER = 0x0032*

• *MILE = 0x0033*

• *KM = 0x0034*

• *YARD = 0x0035*

• *FOOT = 0x0036*

• *DISABLED_PROMPT = 0x0041*

- *AUTOMATIC_PROMPT = 0x0042*

- *MANUAL_PROMPT = 0x0043*

- *CRUISE = 0x0050*

- *MANEUVER_APPEARED = 0x0051*

- *PRE_ADVICE = 0x0052*

- *ADVICE = 0x0053*

- *PASSED = 0x0054*

- *ACTIVE = 0x0060*

- *INACTIVE = 0x0061*

- *STRAIGHT_ON = 0x0070*

- *CROSSROAD = 0x0071*

- *ROUNDABOUT = 0x0072*

- *HIGHWAY_ENTER = 0x0073*

- *HIGHWAY_EXIT = 0x0074*

- *FOLLOW_SPECIFIC_LANE = 0x0075*

- *DESTINATION = 0x0076*

- *WAYPOINT = 0x0077*

- *TURN = 0x0078*

- *BIFURCATION = 0x0079*

- *LEFT = 0x0080*

- *SLIGHT_LEFT = 0x0081*

- *HARD_LEFT = 0x0082*

- *RIGHT = 0x0083*

- *SLIGHT_RIGHT = 0x0084*

- *HARD_RIGHT = 0x0085*

- *UTURN_RIGHT = 0x0086*

- *UTURN_LEFT = 0x0087*

- *ALL_MANUAL = 0x0090*

● *ALL_AUTOMATIC = 0x0091*

● *TRAFFIC_MANUAL = 0x0092*

● *OFF_ROUTE_MANUAL = 0x0093*

● *LATITUDE = 0x00a0*

● *LONGITUDE = 0x00a1*

● *ALTITUDE = 0x00a2*

● *HEADING = 0x00a3*

● *SPEED = 0x00a4*

● *CLIMB = 0x00a5*

● *COUNTRY = 0x00a6*

● *STATE = 0x00a7*

● *CITY = 0x00a8*

● *ZIPCODE = 0x00a9*

● *STREET = 0x00aa*

● *HOUSENUMBER = 0x00ab*

● *CROSSING = 0x00ac*

● *DISTRICT = 0x00ad*

● *PHONENUMBER = 0x00ae*

● *POINAME = 0x00af*

● *TOWNCENTER = 0x00b0*

● *LOCATION_INPUT = 0x00b1*

● *FULL_ADDRESS = 0x00b2*

● *COUNTRYCODE = 0x00b3*

● *HOUSENAME = 0x00b4*

● *NOT_STARTED = 0x00c0*

● *SEARCHING = 0x00c1*

● *FINISHED = 0x00c2*

● *OK = 0x00d0*

- UNKNOWN = 0x00d1

- AMBIGUOUS = 0x00d2

- INCONSISTENT = 0x00d3

- GNSS_FIX_STATUS = 0x00e0

- DR_STATUS = 0x00e1

- MM_STATUS = 0x00e2

- SIMULATION_MODE = 0x00e3

- MATCH_TYPE = 0x00f0

- ON_ROAD = 0x00f1

- OFF_ROAD = 0x00f2

- ON_FERRY = 0x00f3

- IN_TUNNEL = 0x00f4

- ON_CARPARK = 0x00f5

- NO_FIX = 0x0100

- TIME_FIX = 0x0101

- 2D_FIX = 0x0102

- 3D_FIX = 0x0103

- SEGMENT_ID = 0x0110

- DIRECTION_ON_SEGMENT = 0x0112

- DISTANCE_ON_SEGMENT = 0x0113

- INTERMEDIATE_POINTS = 0x0120

- WAYPOINT_TYPE = 0x0121

- SOFT_POINT = 0x0122

- HARD_POINT = 0x0123

- CALCULATION_OK = 0x0130

- NO_POSITION = 0x0131

- UNMATCHED_POSITION = 0x0132

- UNREACHABLE_DESTINATION = 0x0133

● UNFULFILLED_PREFERENCE_MODE = 0x0134

● LINK-ID = 0x0140

● START_LATITUDE = 0x0141

● END_LATITUDE = 0x0142

● START_LONGITUDE = 0x0143

● END_LONGITUDE = 0x0144

● START_ALTITUDE = 0x0145

● END_ALTITUDE = 0x0146

● ROAD_NAME = 0x0147

● DISTANCE = 0x0148

● TIME = 0x0149

● MANEUVER = 0x014a

● INSTRUCTION = 0x014b

● BORDER_CROSSING = 0x014c

● ADDITIONAL_INFORMATION = 0x014d

● ROAD_NUMBER = 0x014e

● START_OFFSET = 0x014f

● FASTEST = 0x0160

● SHORTEST = 0x0161

● ECOLOGICAL = 0x0162

● SCENIC = 0x0163

● EASY = 0x0164

● BALANCED = 0x0166

● CHEAPEST = 0x0167

● FERRY = 0x0170

● TOLL_ROADS = 0x0171

● TUNNELS = 0x0172

● HIGHWAYS_MOTORWAYS = 0x0173

- *VEHICLE_SIZE_LIMIT = 0x0174*

- *CRIME_AREAS = 0x0175*

- *BY_CAR = 0x0180*

- *ON_FOOT = 0x0181*

- *LONG_RANGE_TRAINS = 0x0182*

- *PUBLIC_TRANSPORTATION = 0x0183*

- *BY_BICYCLE = 0x0184*

- *BY_TRUCK = 0x0185*

- *ARRIVAL_TIME = 0x018a*

- *ARRIVAL_DATE = 0x018b*

- *DEPARTURE_TIME = 0x018c*

- *DEPARTURE_DATE = 0x018d*

- *TOTAL_TIME = 0x018e*

- *TOTAL_DISTANCE = 0x018f*

- *PROHIBIT = 0x0190*

- *AVOID = 0x0191*

- *USE = 0x0192*

- *PREFER = 0x0193*

- *IGNORE = 0x0194*

- *TRAFFIC_REALTIME = 0x0200*

- *TRAFFIC = 0x0210*

- *OFF_ROUTE = 0x0211*

- *MANUAL = 0x0212*

- *SIMULATION_STATUS_NO_SIMULATION = 0x0220*

- *SIMULATION_STATUS_RUNNING = 0x0221*

- *SIMULATION_STATUS_PAUSED = 0x0222*

- *SIMULATION_STATUS_FIXED_POSITION = 0x0223*

- *ROAD_FORM_CHANGE = 0x0230*

- *ROAD_REGULAR = 0x0231*

- *ROAD_HIGHWAY_MOTORWAY = 0x0232*

- *ROAD_FERRY = 0x0233*

- *DIRECTION = 0x0240*

- *EXIT_NUMBER = 0x0241*

- *ROAD_FORM = 0x0242*

- *LANE_INFO = 0x0243*

- *LANE_INFO_BITMASK_STRAIGHT = 0x0001*

- *LANE_INFO_BITMASK_SLIGHTRIGHT = 0x0002*

- *LANE_INFO_BITMASK_RIGHT = 0x0004*

- *LANE_INFO_BITMASK_SHARPRIGHT = 0x0008*

- *LANE_INFO_BITMASK_RIGHTUTURN = 0x0010*

- *LANE_INFO_BITMASK_SLIGHTLEFT = 0x0020*

- *LANE_INFO_BITMASK_LEFT = 0x0040*

- *LANE_INFO_BITMASK_SHARPLEFT = 0x0080*

- *LANE_INFO_BITMASK_LEFTUTURN = 0x0100*

- *DIVIDER_UNDEFINED = 0x0250*

- *DIVIDER_INTERRUPTEDLONG = 0x0251*

- *DIVIDER_INTERRUPTEDSHORT = 0x0252*

- *DIVIDER_SOLIDSINGLE = 0x0253*

- *DIVIDER_SOLIDDOUBLE = 0x0254*

- *DIVIDER_SOLIDINTERRUPTED = 0x0255*

- *DIVIDER_INTERRUPTEDSOLID = 0x0256*