

**GENIVI Alliance**

GENIVI Document CS00050

Free Text Search

Component Specification

Draft Version 0.1

**2015-12-22**

**Sponsored by:**

GENIVI Alliance

**Abstract:**

This document provides the Component Specification for the Free Text Search Component. This component provides functionality for searching navigable locations (like addresses and POI's) based on free text.

**Keywords:**

GENIVI, Navigation, Free Text Search, Location Based Services.

**License:**

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

Copyright © , PCA Peugeot Citroën, XS Embedded GmbH, TomTom International B.V., Continental Automotive GmbH, Alpine Electronics R&D Europe GmbH, Harman-Becker Automotive GmbH.  
All rights reserved.

The information within this document is the property of the copyright holders and its use and disclosure are restricted. Elements of GENIVI Alliance specifications may be subject to third party intellectual property rights, including without limitation, patent, copyright or trademark rights (and such third parties may or may not be members of GENIVI Alliance). GENIVI Alliance and the copyright holders are not responsible and shall not be held responsible in any manner for identifying, failing to identify, or for securing proper access to or use of, any or all such third party intellectual property rights.

GENIVI and the GENIVI Logo are trademarks of GENIVI Alliance in the U.S. and/or other countries. Other company, brand and product names referred to in this document may be trademarks that are claimed as the property of their respective owners.

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

The full license text is available at http://creativecommons.org/licenses/by-sa/4.0

The above notice and this paragraph must be included on all copies of this document that are made.

GENIVI Alliance

2400 Camino Ramon, Suite 375

San Ramon, CA 94583, USA

Revision History

Document revision history

|  |  |  |  |
| --- | --- | --- | --- |
| *Date* | *Version* | *Author* | *Description* |
| 2015-12-22 | 0.1 | Peter Goedegebure | Initial revision. |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 Introduction 1](#_Toc438568653)

[1.1 System Overview 1](#_Toc438568654)

[1.2 Subsystem Overview 1](#_Toc438568655)

[1.3 Component Overview 1](#_Toc438568656)

[1.4 Document Overview 2](#_Toc438568657)

[2 References 3](#_Toc438568658)

[3 Glossary 4](#_Toc438568659)

[4 Requirements 5](#_Toc438568660)

[4.1 Use Cases 5](#_Toc438568661)

[4.2 Requirements 7](#_Toc438568662)

[4.2.1 Functional Requirements 7](#_Toc438568663)

[4.2.2 Non-Functional Requirements 8](#_Toc438568664)

[5 Design Constraints 9](#_Toc438568665)

[6 Architecture 10](#_Toc438568666)

[6.1 Architecture Overview 10](#_Toc438568667)

[6.1.1 Component Interfaces 10](#_Toc438568668)

[6.1.2 Component Dependencies 10](#_Toc438568669)

[6.1.3 Component Traceability 10](#_Toc438568670)

[6.1.4 Component Composition 11](#_Toc438568671)

[6.1.5 Component Deployment 11](#_Toc438568672)

[7 Collaboration 12](#_Toc438568673)

[7.1 Use case realization “Finding a location by only typing a search text” 13](#_Toc438568674)

[7.2 Use case realization “Finding an address by only typing a streetname and scrolling down the result list” 14](#_Toc438568675)

[7.3 Use case realization “Finding an address by typing a streetname and indicating that the location is relatively nearby” 16](#_Toc438568676)

[7.4 Use case realization “Finding a hotel by typing its name and selecting the 'hotel' POI category” 17](#_Toc438568677)

[8 Interfaces 19](#_Toc438568678)

[8.1 Interface FreeTextSearch 19](#_Toc438568679)

[8.1.1 Franca IDL definition 19](#_Toc438568680)

[8.1.2 D-Bus XML 32](#_Toc438568681)

[9 Programmer’s Manual 35](#_Toc438568682)

[9.1 RequestId 35](#_Toc438568683)

[10 Implementation 36](#_Toc438568684)

[10.1 Implementation details 36](#_Toc438568685)

[11 Test Plan 37](#_Toc438568686)

[11.1 Test Plan Execution 37](#_Toc438568687)

# Introduction

## System Overview

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

## Subsystem Overview

The Free Text Search Component is part of the Navigation Package as shown in Figure 1.

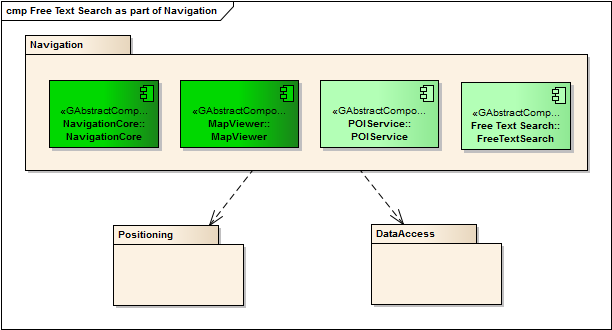


Figure Free Text Search as part of Navigation

Like the POI Service and the LocationInput interface of NavigationCore, the Free Text Search Component provides locations which can e.g. be used as a destination using the Routing interface of NavigationCore.

## Component Overview

Figure 2 shows the context diagram of the component.

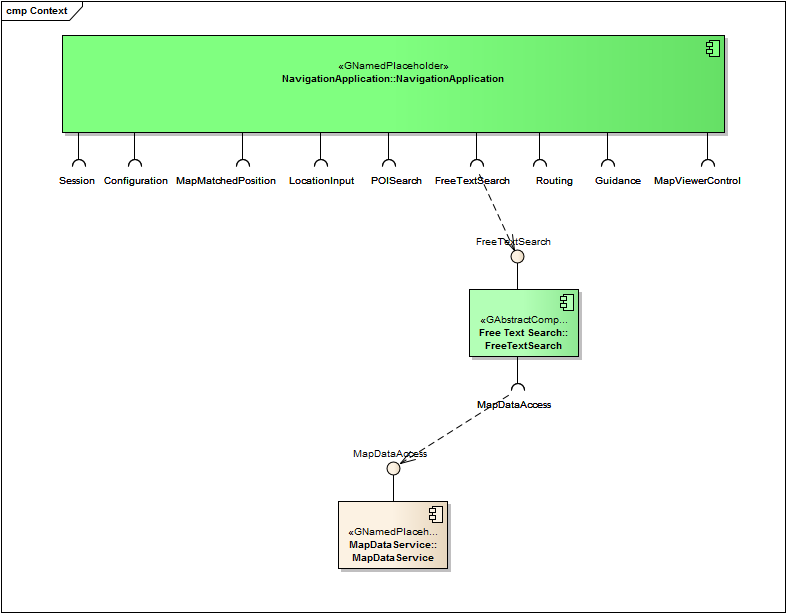


Figure Context diagram of the Free Text Search Component

The Free Text Search component is typically used by a Navigation Application. This Navigation Application uses the FreeTextSearch Interface to let the user search for a location. This location can e.g. be used as destination, waypoint or be shown on the map.

The Free Text Search component depends on the MapDataAccess Interface to obtain the addresses and POIs.

## Document Overview

TODO

*Boiler plate, to be written -- summarize the purpose and contents of this document.*

*Requirements, interface, architecture, test plan, chapters…*

# References

At the time of publication, the referred documents were valid. All standards and specifications are subject to revision, and you are encouraged to investigate the possibility of applying the most recent editions of the standards and specifications indicated below.

1. GENIVI – “GENIVI NavigationCore API” – “3.0.2/30 June 2014” –[http://git.projects.genivi.org/lbs/navigation/doc/navigation-core/NavigationCoreAPI.pdf](http://git.projects.genivi.org/?p=lbs/navigation.git;a=blob_plain;f=doc/navigation-core/NavigationCoreAPI.pdf;hb=HEAD)

# Glossary

|  |  |  |
| --- | --- | --- |
| *Acronym* | *Term* | Definition |
| EA | Enterprise Architect | A proprietary UML tool. EA and UML is widely used in GENIVI interface definitions. |
| FTS | Free Text Search |  |
| UML | Unified Modeling Language |  |

Table – Acronym and Term Definitions

# Requirements

This chapter first lists the Free Text Search related Use Cases, followed by the requirements which are mainly derived from those Use Cases.

## Use Cases

The Use Case diagram for the Free Text Search Use Cases is shown in Figure 1.

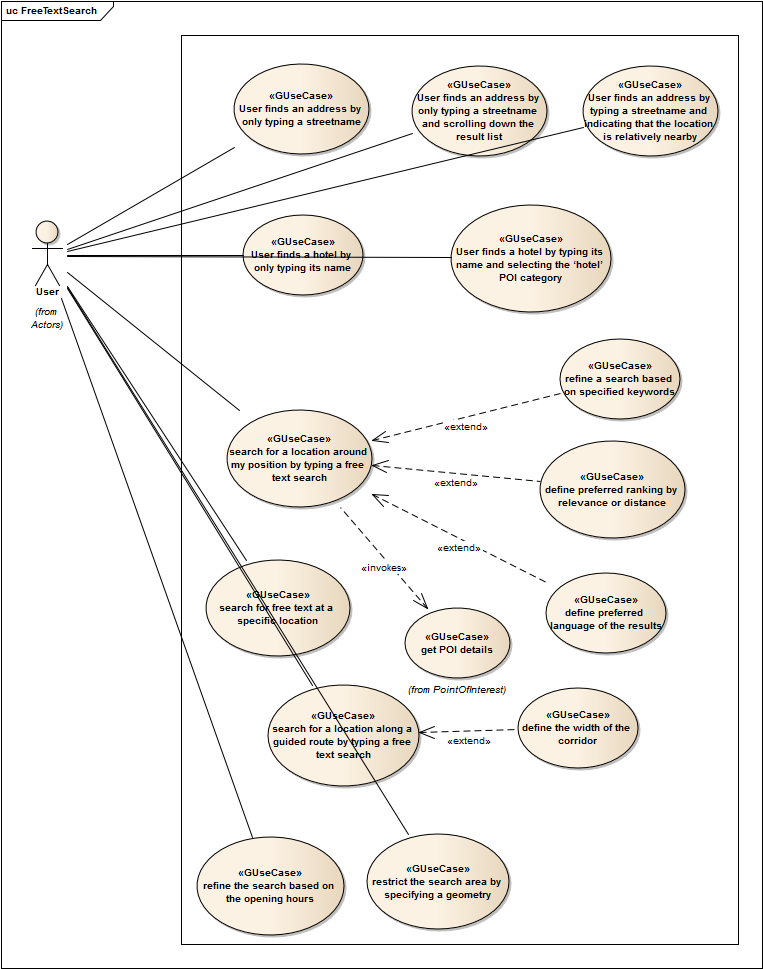


Figure Free Text Search Use Case diagram

*TODO: I’ve defined new clearer Use Cases, these are the 5 Use Cases at the top of the diagram. I propose to delete the existing other ones.*

*Only the 5 new ones are listed in this chapter and are realized.*

The Use Cases are reported in the following format:

|  |  |
| --- | --- |
| **<Use Case ID>** | **<Title>** |
| <Description> | |

The Use Cases show the basic idea of a Free Text Search:

* Independent of the kind of location the use is looking for, he just starts typing a search text
* The systems shows the most relevant locations for this text, and options to narrow down the search
* If the location is shown, the user can directly select it.  
  This is Use Case UC-FTS-010 for addresses and Use Case UC-FTS-013 for POIs.
* If the location is not shown, the user can:
  + Start scrolling until the location appears.  
    This is Use Case UC-FTS-011.
  + Add search criteria to narrow down the search.  
    This is Use Case UC-FTS-012, where ‘nearby’ is added as search criterion, and Use Case UC-FTS-014, where the ‘hotel’ POI category is added as a search criterion.
  + Type more text.  
    There’s no specific Use Case for this.

|  |  |
| --- | --- |
| **UC-FTS-010** | **User finds an address by only typing a streetname** |
| The user types the name of a street in a search text box.  The system lists locations from the complete map database (addresses and POIs), which match the entered text, ordered by relevance. The system also lists relevant POI categories for narrowing down the search.  The street that the user is looking for is in the list, so the user selects it. | |

|  |  |
| --- | --- |
| **UC-FTS-011** | **User finds an address by only typing a streetname and scrolling down the result list** |
| The user types the name of a street in a search text box.  The system lists locations from the complete map database (addresses and POIs), which match the entered text, ordered by relevance. The system also lists relevant POI categories for narrowing down the search.  The street that the user is looking for is not in the list, so the user scrolls down in the list. After scrolling down a few pages, the street that the user is looking for is in the list, so the user selects it. | |

|  |  |
| --- | --- |
| **UC-FTS-012** | **User finds an address by typing a streetname and indicating that the location is relatively nearby** |
| The user types the name of a street in a search text box.  The system lists locations from the complete map database (addresses and POIs), which match the entered text, ordered by relevance. The system also lists relevant POI categories for narrowing down the search.  As the user knows that the street is in a nearby city, he selects the ‘nearby’ option.  The system lists the most relevant locations, based on his location and the entered text, ordered by relevance.  The street that the user is looking for is in the list, so the user selects it. | |

|  |  |
| --- | --- |
| **UC-FTS-013** | **User finds a hotel by only typing its name** |
| The user types the name of a hotel in a search text box.  The system lists locations from the complete map database (addresses and POIs), which match the entered text, ordered by relevance. The system also lists relevant POI categories for narrowing down the search.  The hotel that the user is looking for is in the list, so the user selects it. | |

|  |  |
| --- | --- |
| **UC-FTS-014** | **User finds a hotel by typing its name and selecting the ‘hotel’ POI category** |
| The user types the name of a hotel in a search text box.  The system lists locations from the complete map database (addresses and POIs), which match the entered text, ordered by relevance. The system also lists relevant POI categories for narrowing down the search.  The hotel that the user is looking for is not in the list, and also many other locations (not being hotels) are shown. So the user selects the ‘hotel’ category to narrow down the search. | |

## Requirements

The requirements are reported in the following format:

|  |  |
| --- | --- |
| **<Requirement ID>** | **<Title>** |
| **<Priority>** | *<Components tracing to requirement>* |
| <Description> | |
| *<Rationale>* | |

The fields in the table are linked to the Requirement model element in EA.

### Functional Requirements

The information in this chapter is provided only for information purpose; this is not a normative part. The interface and behavior description (chapter 8 Interfaces) takes precedence in order to fulfill functional requirements.

|  |  |
| --- | --- |
| **SW-NAV-FTS-001** | **Free Text Search** |
| **P1** | <Components tracing to requirement> |
| It shall be possible to retrieve a list of locations by only providing a search text. The returned locations will be the most relevant locations which match the search text. Text matching can be done on any combination of the properties of the location like e.g. city name, street name, house number, POI name, POI category. The definition of most relevant is up to the implementation. | |
| This functionality is needed to support an HMI with a one-box-search, i.e. where the user can enter a text in a single text field, instead of having to choose for an address or POI first and then either enter country, city, street and house number, or a POI category. | |

|  |  |
| --- | --- |
| **SW-NAV-FTS-002** | **Free Text Search around location** |
| **P1** | <Components tracing to requirement> |
| It shall be possible to specify a search location in combination with a search text. In this case locations closer to the search location are more relevant then locations further away from the search location. | |
| The user is often searching for locations relatively near his location. By using both a search text and a location, a shorter text is needed to find the wanted location. Example: The user is looking for a street in the current city. By setting the search location to his current location, usually just a few characters are needed to find the street. | |

|  |  |
| --- | --- |
| **SW-NAV-FTS-003** | **Free Text Search restricted to specific areas** |
| **P1** | <Components tracing to requirement> |
| It shall be possible to restrict a Free Text Search to specific areas. | |
| If the locations are to be shown on a map, it makes no sense to retrieve location outside of the shown map area. | |

|  |  |
| --- | --- |
| **SW-NAV-FTS-004** | **Free Text Search restricted to one or more POI categories** |
| **P1** | <Components tracing to requirement> |
| It shall be possible to restrict a Free Text Search to one or more POI categories. For this, an initial Free Text Search will provide a list of relevant POI categories. | |
| By using both a search text and one or more POI categories, a shorter text is sufficient to find a POI location. | |

|  |  |
| --- | --- |
| **SW-NAV-FTS-005** | **Free Text Search restricted to one or more countries or states** |
| **P1** | <Components tracing to requirement> |
| It shall be possible to restrict a Free Text Search to one or more countries or states. | |
| By using both a search text and one or more countries,or states, a shorter text is sufficient to find a location. Also, there are often many reasons to look for a (POI) location at a specific side of a border (e.g. passport, currency, insurance restrictions). | |

### Non-Functional Requirements

This is a normative part. These requirements shall be fulfilled by compliant implementations.

There are no non-functional requirements.

# Design Constraints

This is a normative part that includes additional requirements on a compliant implementation outside of the interface and behavior descriptions.

There are no Design Constraints.

# Architecture

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

As the Free Text Search Component is an Abstract Component, this chapter only contains information related to the outside of the component.

## Architecture Overview

The architecture overview is shown in Figure 4.

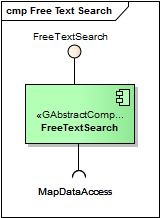


Figure 4 Free Text Search Component

### Component Interfaces

The component provides the following interface:

* FreeTextSearch  
  This interface provides functionality for searching for locations, based on text and optional other criteria to narrow down the search.

### Component Dependencies

The component requires the following interface:

* MapDataAccess  
  MapDataAccess provides access to the actual locations in the map database. The map database shall contain indices to support an efficient search for these locations.

### Component Traceability

Figure 5 shows the requirements traceability, i.e. which requirements (as listed in chapter 4) are implemented by the component.

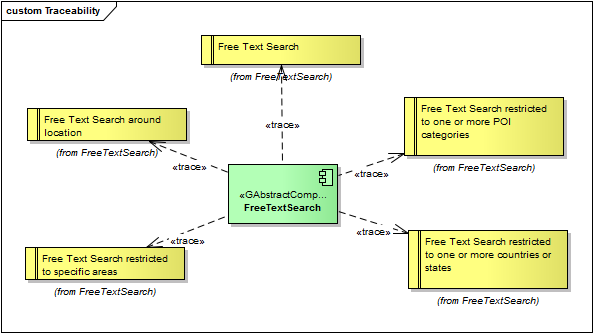


Figure Requirements traceability

### Component Composition

Not applicable as this is an Abstract Component.

### Component Deployment

Not applicable as this is an Abstract Component.

# Collaboration

This is a normative part of the document.

The Use Case realizations are shown in Figure 6.

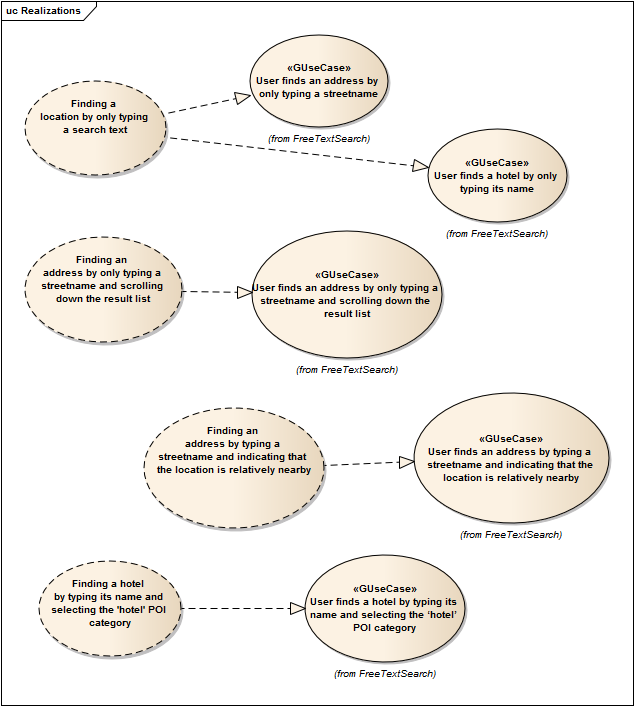


Figure Free Text Search Use Case realizations

Overview of the Use Case realizations:

* “User finds an address by only typing a streetname” and “User finds a hotel by only typing its name”  
  realized by “Finding a location by only typing a search text”  
  The fact that these two Use Cases are realized by one Use Case Realization, shows that Free Text Search basically works the same for all types of locations.
* “User finds an address by only typing a streetname and scrolling down the result list”  
  realized by “Finding an address by only typing a streetname and scrolling down the result list”  
  This realization uses the paging mechanism of the interface.  
  The Use Case is about finding an address, but the realization also applies to POIs.
* “User finds an address by typing a streetname and indicating that the location is relatively nearby”  
  realized by “Finding an address by typing a streetname and indicating that the location is relatively nearby”  
  This realization shows the use of a searchLocation to narrow down the search.  
  The Use Case is about finding an address, but the realization also applies to POIs.
* “User finds a hotel by typing its name and selecting the ‘hotel’ POI category”  
  realized by “Finding a hotel by typing its name and selecting the 'hotel' POI category”  
  This realization shows the use of a searchOption to narrow down the search.  
  This option is only applicable for POIs.

## Use case realization “Finding a location by only typing a search text”

The sequence diagram for this Use Case realization is shown in Figure 7.

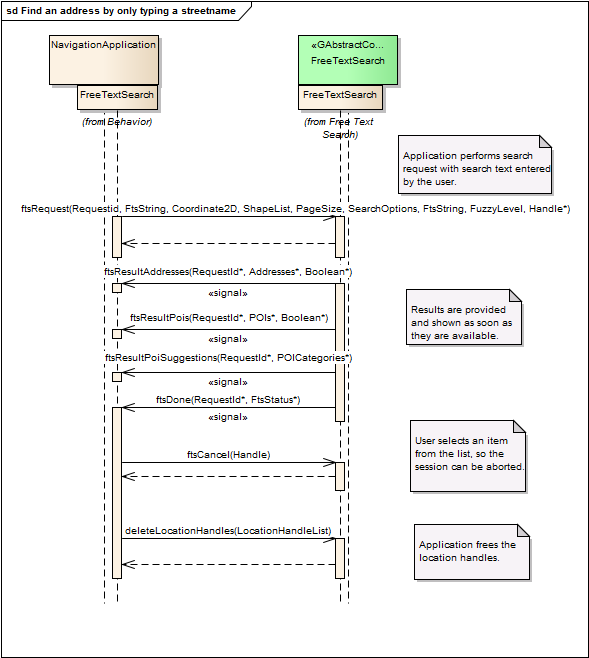


Figure Sequence Diagram - Finding a location by only typing a search text

It’s up to the NavigationApplication to decide when to start the search. This can e.g. be as soon as the user entered a specific number of characters, or when the user presses a ‘search’ button.

In this example, there are results for addresses, POIs and POI category suggestions. Note that the order in which these results are provided is not specified.

Each returned location has a LocationHandle. When such a handle is no longer needed, the application has to free it by calling deleteLocationHandles().

## Use case realization “Finding an address by only typing a streetname and scrolling down the result list”

The sequence diagram for this Use Case realization is shown in Figure 8.

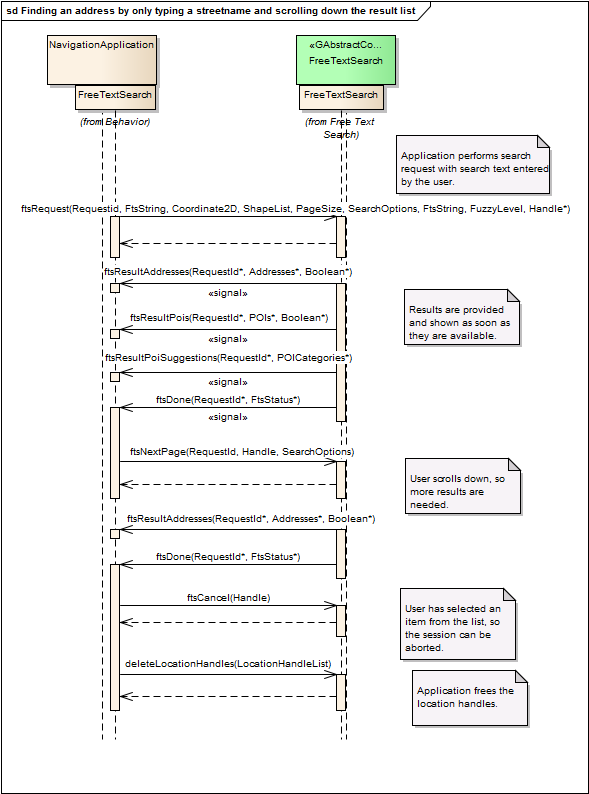


Figure Sequence Diagram - Finding an address by only typing a street name and scrolling down the result list

How a user can scroll through the list of results is of course application specific, but typically this per page, or by one or more lines. If it is by page, the pageSize could be equal to the number of locations that can be shown on the screen. If it is by line, the pageSize could be somewhat larger than the number of locations that can be shown on the screen.  
In any case, when the user starts scrolling, at some point more results are needed. So the application call ftsNextPage(). In this example no more POI results are available, so only a next set of address is received.

Of course the user can scroll back, so the application will remember all the results, until the user has selected one. Therefore there’s only a call to deleteLocationHandles() at the end of the sequence.

## Use case realization “Finding an address by typing a streetname and indicating that the location is relatively nearby”

The sequence diagram for this Use Case realization is shown in Figure 9.

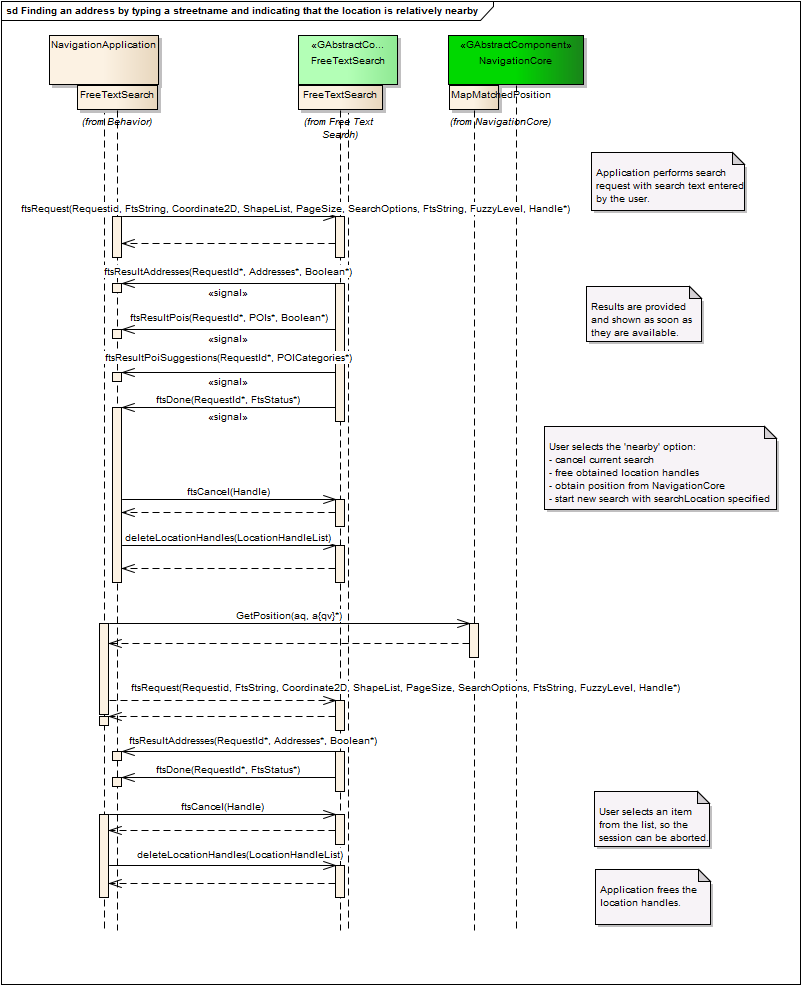


Figure Sequence Diagram - Finding an address by typing a street name and indicating that the location is relatively nearby

The user starts a search with only a search text. Based on the results he decides to add the ‘near’ option.  
This means that a new search will be started, with an extra parameter ‘searchLocation’So:

* The current search has to be stopped, so ftsCancel() is called.
* The current results will no longer be used, so the location handles can be freed by calling deleteLocationHandles().
* The current location is obtained from NavigationCore.
* A new ftsRequest() is issued with searchLocation set to the current location.

In this example the second search only returns address, but it can of course also return POIs and POI Category suggestions.

It is of course also possible that the user selects the ‘near’ option directly. In this case the sequence would start with the GetPosition() call.

## Use case realization “Finding a hotel by typing its name and selecting the 'hotel' POI category”

The sequence diagram for this Use Case realization is shown in Figure 10.

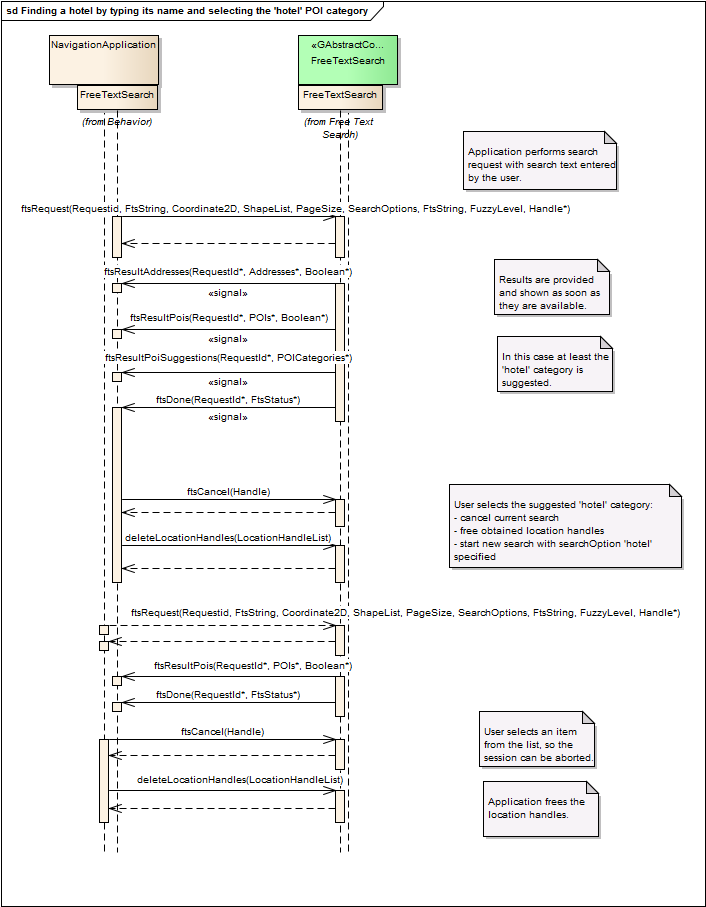


Figure Sequence Diagram - Finding a hotel by typing its name and selecting the 'hotel' POI category

The user starts a search with only a search text. In this case the ‘hotel’ category is returned (and shown by the application) as one of the POI Category suggestions. The user selects this category to narrow down his search. So:

* The current search has to be stopped, so ftsCancel() is called.
* The current results will no longer be used, so the location handles can be freed by calling deleteLocationHandles().
* A new ftsRequest() is issued with searchOptions set to the ‘hotel’ category.  
  Also via the searchOptions it will be indicated that only POI should be returned.

# Interfaces

This is a normative part of the document.

## Interface FreeTextSearch

### Franca IDL definition

The Franca IDL definition is based on Franca 0.9.1.

// SPDX-License-Identifier: MPL-2.0

// Copyright (C) 2015-2016, PCA Peugeot Citroën, XS Embedded GmbH, TomTom International B.V., Continental Automotive GmbH,

// Alpine Electronics R&D Europe GmbH, Harman-Becker Automotive GmbH.

// This Source Code Form is subject to the terms of the

// Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with

// this file, you can obtain one at http://mozilla.org/MPL/2.0/.

package org.genivi.navigation.freetextsearchservice

import org.genivi.navigation.NavigationTypes.\* from "../NavigationTypes.fidl"

<\*\* @description : This interface allows a user to find locations by entering a single text string.

A user of a navigation system needs to find a location on a map, e.g. to use as a destination of a route.

Locations can be identified by different means, for example:

- An address, e.g. '2400 Camino Ramon, San Ramon, California, USA.'

- A point of interest, e.g. 'Eiffel Tower'.

- A named place, .e.g. 'Amsterdam'

This interface allows a user to find these locations by entering a single text string.

The system will respond with locations that match the given text string.

For each location, a location handle will be returned that can be used as input to other interfaces,

for example to plan a route, or to get more attributes of a point of interest.

A free text search is initiated by sending a free text search request (Method FtsRequest) containing the search text and search options input by the user.

Depending on search options both addresses and POIs are searched for matches. A single request may get multiple result responses (via Signals):

- FtsResultAddresses to report address matches

- FtsResultPois to report POI matches

- FtsResultPoiSuggestions to give suggestions for doing specific POI queries

Each response comes as soon as the results are available, so address results may be received while the search process continues looking for POI matches.

The order in which the results will be received is not defined. The search engine determines which results to return first based on the user input.

If there are no matches found in a certain type of result, then no response for that type will be sent.

E.g. if the user input only matches an address but not POIs, then no FtsResultPois response will be sent.

To indicate that the search has finished an FtsDone signal is sent.

\*\*>

interface FreeTextSearch {

version {

major 0

minor 1

}

<\*\* @description: A requestId is an identifier used to match a response to a request.

Range[0x1:0x7fff]

notSpecifiedValue = 0x0

\*\*>

typedef RequestId is Int16

<\*\* @description: Handle for a single Free Text Search session.

\*\*>

typedef FreeTextSearchHandle is Handle

<\*\* @description: Type to represent most texts in this interface..

Character set = UNICODE

Maximum length = 4095

\*\*>

typedef FtsString is String

<\*\* @description: A list of strings

maximum length = 100

\*\*>

array FtsStringList of FtsString

<\*\* @description: A union to contain any of the possble search shapes. \*\*>

union Shape {

Circle circle

Rectangle rectangle

}

<\*\* @description: A list of shapes.

maximum length = 31

\*\*>

array ShapeList of Shape

<\*\* @description: A maximum number of results to be returned.

Range[0:1000]

\*\*>

typedef PageSize is UInt16

<\*\* @description: Search options for a search request.

The options are flags which can be OR-ed together to create SearchOptions.

\*\*>

enumeration SearchOption {

<\*\* @description: Search addresses. \*\*>

ADDRESS = 1

<\*\* @description: Search POIs. \*\*>

POI = 2

<\*\* @description: Get POI suggestions. \*\*>

POI\_SUGGESTION = 4

<\*\* @description: Do not search places when doing an address search. \*\*>

NO\_PLACES = 65536

<\*\* @description: Do not search streets when doing an address search. \*\*>

NO\_STREETS = 131072

<\*\* @description: Do not make assumptions on POI categories. \*\*>

NO\_POI\_CATEGORY\_ASSUMPTIONS = 262144

}

<\*\* @description: An OR-ed combination of SearchOption flags.

Range[0:0x7fffffff]

\*\*>

typedef SearchOptions is UInt32

<\*\* @description: Search request status. \*\*>

enumeration FtsStatus {

OK

<\*\* @description: User aborted search. \*\*>

ABORTED

<\*\* @description: Search string is too short to evaluate. \*\*>

PREFIX\_TOO\_SHORT

<\*\* @description: There is no search for which a next page can be requested. \*\*>

NO\_SEARCH\_TO\_CONTINUE

INTERNAL\_ERROR

<\*\* @description: No FTS index for the current map. \*\*>

INDEX\_MISSING

<\*\* @description: The FTS index format is not supported. \*\*>

BAD\_VERSION

<\*\* @description: A search query not supported by the FTS engine. \*\*>

INVALID\_PARAMETER\_QUERY

<\*\* @description: A search location not on this earth. \*\*>

INVALID\_PARAMETER\_SEARCH\_LOCATION

<\*\* @description: An invalid page size. \*\*>

INVALID\_PARAMETER\_PAGE\_SIZE

<\*\* @description: Invalid search options passed to the FTS engine. \*\*>

INVALID\_PARAMETER\_SEARCH\_OPTIONS

<\*\* @description: Invalid search conditions passed to the FTS engine. \*\*>

INVALID\_PARAMETER\_SEARCH\_CONDITIONS

<\*\* @description: One or more of the search shapes are invalid. \*\*>

INVALID\_PARAMETER\_SEARCH\_SHAPES

}

<\*\* @description: Handle to a location.

Range[0:0x7fffffff]

\*\*>

typedef LocationHandle is UInt32

<\*\* @description: A list of LocationHandles.

maximum length = 4096

\*\*>

array LocationHandleList of LocationHandle

<\*\* @description: If a user entered a non-existing house number then the closest existing

number is returned. The houseNumberFromInput will contain the house

number from the user input, the houseNumber will contain closest house number.

If the map does not contain house numbers for the street, but the user

input seems to have a house number, then houseNumber will be empty and

houseNumberFromInput will contain the number from the input.

\*\*>

struct StreetDetails {

FtsString streetName

<\*\* @description: If a user entered a non-existing house number then the closest existing

number is returned. The houseNumberFromInput will contain the house

number from the user input, the houseNumber will contain closest house number.

If the map does not contain house numbers for the street, but the user

input seems to have a house number, then houseNumber will be empty and

houseNumberFromInput will contain the number from the input.

\*\*>

FtsString houseNumber

<\*\* @description: The house number that the search engine recognized in the user input.

Can be empty.

\*\*>

FtsString houseNumberFromInput

}

enumeration PlaceType {

<\*\* @description: a place of unknown type. \*\*>

OTHER

<\*\* @description: a center of a settlement. \*\*>

SETTLEMENT = 2

}

<\*\* @description: Place details. \*\*>

struct PlaceDetails {

PlaceType placeType

FtsString placeName

<\*\* @description: The bounding box of this place (an estimate). \*\*>

Rectangle placeBoundingBox

}

<\*\* @description: Mapcode details. \*\*>

struct MapCodeDetails {

<\*\* @description: If the location of the mapcode is close to a street, this field

contains the name of that street, otherwise this field is left empty.

\*\*>

FtsString closestStreetName

}

<\*\* @description: A union to contain any of the specific types of address details. \*\*>

union AddressDetails {

StreetDetails streetDetails

PlaceDetails placeDetails

MapCodeDetails mapCodeDetails

}

<\*\* @description: A higher score means a better match.

Range[0:0x7fff]

\*\*>

typedef Score is Int16

<\*\* @description: Address information. \*\*>

struct Address {

LocationHandle locationHandle

FtsString countryCode

<\*\* @description: Optional state code. \*\*>

FtsString stateCode

<\*\* @description: There can be multiple place names. \*\*>

FtsStringList places

<\*\* @description: There can be multiple postal codes assigned to a street. \*\*>

FtsStringList postalCodes

<\*\* @description: Mapcode of the coordinate \*\*>

FtsString mapCode

AddressDetails addressDetails

Coordinate2D coordinate

<\*\* @description: Distance in meters from the search location.

-1 if no search location is given.

\*\*>

DistanceInMeters distance

<\*\* @description: A higher score means a better match. \*\*>

Score score

<\*\* @description: Indicates whether this match is fuzzy, i.e. it was matched with misspellings. \*\*>

Boolean fuzzyMatch

}

<\*\* @description: A list of addresses.

maximum length = 500

\*\*>

array Addresses of Address

<\*\* @description: Numerical POI category code.

Range[0:0xffffffff]

\*\*>

typedef POICategoryCode is UInt32

<\*\* @description: POI information. \*\*>

struct POI {

LocationHandle locationHandle

POICategoryCode categoryCode

FtsStringList brandNames

FtsString poiName

FtsString address

FtsStringList place

FtsStringList postalCode

FtsString mapCode

FtsString countryCode

FtsString stateCode

FtsString telephone

Coordinate2D coordinate

<\*\* @description: Distance in meters from the search location.

-1 if no search location is given.

\*\*>

DistanceInMeters distance

<\*\* @description: A higher score means a better match. \*\*>

Score score

<\*\* @description: Indicates whether this match is fuzzy, i.e. it was matched with misspellings. \*\*>

Boolean fuzzyMatch

}

<\*\* @description: A list of POIs.

maximum length = 500

\*\*>

array POIs of POI

<\*\* @description: Data type for POI category suggestions. \*\*>

struct POICategory {

POICategoryCode categoryCode

FtsString categoryName

<\*\* @description: The search engine recognizes synonyms and names associated with a

POI category. If a match on such a name is found, then this attribute is set to that name.

\*\*>

FtsString matchedName

<\*\* @description: Text that can be used for a suggestion like: "<categoryName> <searchTextSuggestion>"

Typically this contains the original query text, where the recognized

category name or synonym has been removed.

\*\*>

FtsString searchTextSuggestion

<\*\* @description: A list of addresses that can be used for suggestions like: "<categoryName> near <address>

Note the addresses in this list do not have a valid location handle

as the address is only needed to be displayed as a suggestion.

The coordinates of an address can be used for a POI window query if the

user picks a suggestion.

\*\*>

Addresses nearAddressSuggestions

<\*\* @description: A higher score means a better match. \*\*>

Score score

}

<\*\* @description: A list of POI categories.

maximum length = 500

\*\*>

array POICategories of POICategory

<\*\* @description: Indicates on a scale from 0 to 5 how fuzzy the search results can be.

0 means only an exact search is performed, 1 is the lowest level of fuzzy search

and 5 means a very fuzzy search.

Range[0:5]

\*\*>

typedef FuzzyLevel is Int8

<\*\* @description : This method returns the API version implemented by the content access module.\*\*>

method getVersion {

out {

<\*\* @description: .\*\*>

Version ^version

}

}

<\*\* @description: Perform a free text search.

A single search request will trigger zero or more FtsResultXxx responses followed by an FtsDone response.

Each response will contain 1 page of results.

A next page of results can be retrieved with the FtsNextPage request.

When you send a new request before receiving a done response on the previous request,

then the previous request will be canceled.

You will still get a done response on the previous request.

This way each request gets a response.

\*\*>

method ftsRequest {

in {

<\*\* @description: Identifier to match responses to this request. \*\*>

RequestId requestId

<\*\* @description: Text to match. \*\*>

FtsString inputString

<\*\* @description: Location for scoring the search results.

A result closer to this location gets a higher score than a result further away.

\*\*>

Coordinate2D searchLocation

<\*\* @description: Limit the search to the union of several shapes (each can be a circle or a rectangle).

Pass an empty list to search the whole map.

\*\*>

ShapeList searchShapes

<\*\* @description: The maximum number of addresses and POIs that will be returned. \*\*>

PageSize pageSize

<\*\* @description: Options to indicate what to search for.

One or more values of SearchOption can be added together.

\*\*>

SearchOptions searchOptions

<\*\* @description: Additional search conditions, space separated.

The string can be empty when the POI search is not limited by categories

and not limited by countries/states.

Supported conditions:

"category:nnnn" Limit the POI search to a one or more POI categories,

where 'nnnn' is a comma-separated list of numerical POI category ids.

"country:aaaaa" Limit the search to one or more countries (or states),

where 'aaaaa' is a comma-separated list of ISO-3166-1 alpha 3

country code concatenated with an optional ISO-3166-2 state code.

Examples:

"category:9373,9374": limits the POI search to the union of categories 9373 and 9374.

"country:USACA": limits the search to the state of California.

"category:6315 country:USACA,MEX": limits the POI search to category 6315

and limits the search to the union of California state and Mexico.

\*\*>

FtsString searchConditions

<\*\* @description: Indication of how fuzzy the search shall be performed. \*\*>

FuzzyLevel fuzzyLevel

}

out {

<\*\* @description: Identifier to match this response with a request. \*\*>

RequestId responseId

<\*\* @description: Handle to identify the Free Text Search session.

\*\*>

Handle freeTextSearchHandle

}

error {

<\*\* @description: Indicates that no new Free Text Search session can be started at the moment.

\*\*>

NoMoreFtsHandles

}

}

<\*\* @description: Get next result page for the last performed search.

This request will trigger zero or more FtsResultXxx responses followed by an FtsDone response.

If there is no search request done before or if a search request is still in progress

hen the FtsDone response will return a EStatusNoSearchToContinue status.

\*\*>

method ftsNextPage {

in {

<\*\* @description: Identifier to match responses to this request. \*\*>

RequestId requestId

<\*\* @description: Handle to identify the Free Text Search session.

\*\*>

Handle freeTextSearchHandle

<\*\* @description: Options to indicate what to search for.

One or more values of TFTSSearchOption can be added together.

You can only get results for options that have been passed to the initial FtsRequest.

\*\*>

SearchOptions searchOptions

}

}

<\*\* @description: Cancel an FTS session. \*\*>

method ftsCancel {

in {

<\*\* @description: Handle to identify the Free Text Search session.

\*\*>

Handle freeTextSearchHandle

}

}

<\*\* @description : Response indicating that the search is done. \*\*>

broadcast ftsDone selective {

out {

<\*\* @description: Identifier to match this response with a request. \*\*>

RequestId requestId

<\*\* @description: Status of the search request. \*\*>

FtsStatus ftsStatus

}

}

<\*\* @description : Response with address results. \*\*>

broadcast ftsResultAddresses selective {

out {

<\*\* @description: Identifier to match this response with a request. \*\*>

RequestId requestId

<\*\* @description: Matching addresses.

The order of the addresses is the order of matching relevance.

The first address is a better match than the last result.

\*\*>

Addresses addresses

<\*\* @description: Flag to indicate if more addresses might be available in a next page. \*\*>

Boolean moreAvailable

}

}

<\*\* @description : Response with POI results. \*\*>

broadcast ftsResultPois selective {

out {

<\*\* @description: Identifier to match this response with a request. \*\*>

RequestId requestId

<\*\* @description: Matching POIs.

The order of the POIs is the order of matching relevance.

The first POI is a better match than the last result.

\*\*>

POIs pois

<\*\* @description: Flag to indicate if more POIs might be available in a next page. \*\*>

Boolean moreAvailable

}

}

<\*\* @description : Response with POI category suggestions. \*\*>

broadcast ftsResultPoiSuggestions selective {

out {

<\*\* @description: Identifier to match this response with a request. \*\*>

RequestId requestId

<\*\* @description: POI category suggestions. \*\*>

POICategories poiCategories

}

}

<\*\* @description : Release handle(s) if they are not used anymore. \*\*>

method deleteLocationHandles {

in {

<\*\* @description: list of handles to release. \*\*>

LocationHandleList locationHandleList

}

}

}

### D-Bus XML

This D-Bus XML interface definition is generated from the Franca IDL definition with the use of CommonAPI version 3.1.3.

<?xml version="1.0" encoding="UTF-8"?>

<!-- SPDX-License-Identifier: MPL-2.0

Copyright (C) 2015-2016, PCA Peugeot Citroën, XS Embedded GmbH, TomTom International B.V., Continental Automotive GmbH,

Alpine Electronics R&D Europe GmbH, Harman-Becker Automotive GmbH.

This Source Code Form is subject to the terms of the

Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with

this file, you can obtain one at http://mozilla.org/MPL/2.0/.

-->

<node xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="introspect.xsd" name="org.genivi.navigation.freetextsearchservice">

<interface name="FreeTextSearch">

<version>0.1</version>

<method name="getInterfaceVersion">

<arg name="value" type="uu" direction="out" /></method>

<signal name="FtsDone">

<arg name="requestId" type="n" />

<arg name="ftsStatus" type="i" />

</signal>

<signal name="FtsResultAddresses">

<arg name="requestId" type="n" />

<arg name="addresses" type="a(ussasass(yv)(dd)inb)" />

<arg name="moreAvailable" type="b" />

</signal>

<signal name="FtsResultPois">

<arg name="requestId" type="n" />

<arg name="pois" type="a(uuasssasasssss(dd)inb)" />

<arg name="moreAvailable" type="b" />

</signal>

<signal name="FtsResultPoiSuggestions">

<arg name="requestId" type="n" />

<arg name="poiCategories" type="a(usssa(ussasass(yv)(dd)inb)n)" />

</signal>

<method name="getVersion">

<arg name="\_version" type="(qqqs)" direction="out" />

</method>

<method name="FtsRequest">

<arg name="\_requestId" type="n" direction="in" />

<arg name="\_inputString" type="s" direction="in" />

<arg name="\_searchLocation" type="(dd)" direction="in" />

<arg name="\_searchShapes" type="a(yv)" direction="in" />

<arg name="\_pageSize" type="q" direction="in" />

<arg name="\_searchOptions" type="u" direction="in" />

<arg name="\_searchConditions" type="s" direction="in" />

<arg name="\_fuzzyLevel" type="y" direction="in" />

<arg name="\_error" type="i" direction="out" />

<arg name="\_freeTextSearchHandle" type="u" direction="out" />

</method>

<method name="FtsNextPage">

<arg name="\_requestId" type="n" direction="in" />

<arg name="\_freeTextSearchHandle" type="u" direction="in" />

<arg name="\_searchOptions" type="u" direction="in" />

</method>

<method name="FtsCancel">

<arg name="\_freeTextSearchHandle" type="u" direction="in" />

</method>

<method name="DeleteLocationHandles">

<arg name="\_locationHandleList" type="au" direction="in" />

</method>

</interface>

</node>

# Programmer’s Manual

This is an optional non-normative part providing examples of how to program against the interfaces.

## RequestId

In asynchronous interfaces it is good practice to always use a mechanism for matching a response to a specific request. The requestId is such a mechanism.

Each request has a requestId parameter. So the requestId is ‘generated’ by the client.  
Each response also has a requestId parameter, where the value is the value of the requestId of the request to which this is a response.

The client typically uses a simple counter to generate requestIds.

Example without a requestId:

* Client issues request A
* Client issues request B
* Client receives a response  
  Is this a response to request A or to request B?  
  Mostly it will be for A, but request A or its response may have got lost. Or processing of request A takes longer the for request B.

Example with a requestId:

* Client issues request A, requestId is 1.
* Client issues request B, requestId is 2.
* Client receives a response, requestId is 1  
  This is clearly the response to request A.

# Implementation

TODO

This section describes the implementation of the **proof-of-concept** for the FreeTextSearch component.

The following implementation was used for proving the specification. Note that updated versions of this software may exist:

*Place a link to the specific tagged version in the repository (or similar unambiguous version information).*

## Implementation details

*Place a link to a separate (public) document containing a description of the available implementation.   
Referencing a good quality README could fulfill a large part of this.*

*It should include:*

* *General description of how to build the software.*
* *Link to the source code repository.*
* *Diagrams or other visual explanation.*
* *Code examples that are specific to the PoC/RI.*

*Example: The PoC was built using C++ and Qt, under Ubuntu 13.04, with GCC 4.2.1   
The source code can be found at git://mygitrepo/mytag (substitute this with a field)  
etc.*

# Test Plan

TODO

A Test Plan specifies the test scenarios for the Abstract Component that can be executed against its Proof of Concept or Reference Implementation. The purpose of the Test Plan is to validate the component APIs. It is not required that the execution of the Test Plan is automated, but it can be.

*The acceptable ways to provide a Test Plan include (a) a textual description of the test sequences and their results; (b) UML sequence diagrams; (c) automated and executable test code against the Proof of Concept or Reference Implementation.*

*Document the test plan description or link to a separate document and/or executable test code.*

*In particular if the test plan contains a long list of test cases, it is better suited as a separate document. Describing the strategy for testing can however be suitable inside of this document.*

## Test Plan Execution

A compliant implementation of this abstract component specification must execute the test plan successfully and document the results.