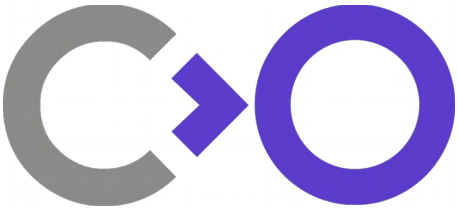

Apertis Geolocation and Navigation Design

Author:	Philip Withnall
Contributors:	Guillaume Desmottes, Simon McVittie
Version:	0.2.1
Status:	Draft
Date:	2015-11-12
Last Reviewer:	Simon McVittie

This design was produced exclusively using free and open source software.

Please consider the environment before printing this document.

DOCUMENT CHANGE LOG

Version	Date	Changes
0.2.1	2015-11-12	• Clarified application activation on receipt of signals • Clarify terminology ‘signal’ vs ‘notification’
0.2.0	2015-11-04	• Added more use cases and reworked requirements • Clarified requirements for multiple backends
0.1.2	2015-10-21	• All navigation functionality must be available when the vehicle is offline.
0.1.1	2015-10-14	• Clarification of open questions about old APIs.
0.1.0	2015-10-14	• New document to summarise background research.

Table of Contents

Document Change Log 2
1 Introduction 6
2 Terminology and concepts 7
2.1 Coordinates 7
2.2 Geolocation 7
2.3 Forward geocoding 7
2.4 Reverse geocoding 7
2.5 Geofencing 7
2.6 Route planning 7
2.7 Route replanning 7
2.8 Route cancellation 7
2.9 Point of interest 8
2.10 Text-to-speech (TTS) 8
3 Use cases 9
3.1 Relocating the vehicle 9
3.2 Automotive backend 9
3.2.1 Custom automotive backend functionality 9
3.3 SDK backend 9
3.4 Different types of vehicle 9
3.5 Route calculation for public transport 9
3.6 Audio-only route guidance 10
3.7 Viewing an address on the map 10
3.8 Finding the address of a location on the map 10
3.9 Type-ahead search and completion for addresses 10
3.10 Navigating between two places 10
3.11 Changing destination during navigation 11
3.12 Navigating via waypoints 11
3.13 Estimated time left in a navigation 11
3.14 Deviating off a navigation route 11
3.15 Changing navigation route due to traffic conditions 11
3.16 Turn-by-turn navigation directions 12
3.17 Navigation road information 12
3.18 Resuming a navigation route 12
3.19 Cancelling a navigation route 12
3.20 Tasks nearby 12
3.21 Turning on house lights 12
3.22 Travelling abroad 12
3.23 Temporary loss of GPS signal 13
3.24 Look up the address of a contact 13
3.25 Navigation-driven refinement of geolocation 13
3.26 Application-driven refinement of geolocation 13
3.26.1 Excessive results from application-driven refinement of geolocation 13

3.27 Malware application bundle....13
4 Non-use-cases....14
4.1 POI data....14
4.2 Beacons....14
4.3 Loading map tiles from a backend....14
5 Requirements....15
5.1 Geolocation API....15
5.2 Geolocation service supports signals....15
5.3 Geolocation implements caching....15
5.4 Geolocation supports backends....15
5.5 Navigation routing API....15
5.6 Navigation routing supports backends....16
5.7 Type-ahead search and address completion supports backends....16
5.8 Geocoding supports backends....16
5.9 SDK has default implementations for all backends....16
5.10 SDK APIs do not vary with backend....16
5.11 Navigation routing supports vehicle types....16
5.12 Navigation routing supports alternative routes....17
5.13 Turn-by-turn instructions are suitable for TTS....17
5.14 TTS route guidance must be implemented as a service....17
5.15 2D map rendering API....17
5.16 Reverse geocoding API....17
5.17 Forward geocoding API....18
5.18 Type-ahead search and address completion API....18
5.19 Navigation routing supports waypoints....18
5.20 Navigation routing supports different optimisations....18
5.21 Navigation routing provides turn-by-turn instructions....18
5.22 3D map rendering API....19
5.23 Navigation routing provides statistics....19
5.24 Navigation routing provides road information....19
5.25 Navigation routing can wake up applications....19
5.26 Geofencing API....19
5.27 Geofencing service can wake up applications....20
5.28 Navigation routing must be locale-aware....20
5.29 Geocoding API must be locale-aware....20
5.30 Geolocation provides error bounds....20
5.31 Geolocation implements dead-reckoning....20
5.32 Geolocation uses navigation data if available....20
5.33 Geocoding uses general points of interest streams....20
5.34 Location information requires permissions to access....21
5.35 Rate limiting of general point of interest streams....21
6 Existing geo systems....22
6.1 W3C Geolocation API....22
6.2 Android platform location API....22
6.3 Google Location Services API for Android....22
6.4 iOS Location and Maps API....22

6.5 GNOME APIs....23
6.5.1 GeoClue....23
6.5.2 Geocode-glib....24
6.5.3 libchamplain....24
6.5.4 NavIt....24
6.6 Navigation routing systems....24
6.6.1 GraphHopper....24
6.6.2 OSRM....25
6.6.3 YOURS....25
6.7 NavServer....25
7 Approach....26
7.1 Backends....26
7.2 2D map display....26
7.3 3D map display....27
7.4 Geolocation....27
7.5 Geolocation signals....27
7.6 Navigation routing....28
7.7 Localisation of geocoding....28
7.8 Localisation of routing information....29
7.9 Forward and reverse geocoding....29
7.10 Address completion....29
7.11 Route alerts....30
7.12 Geofencing....30
7.13 Location security....30
7.14 Systemic security....31
7.15 Requirements....31
7.16 Suggested roadmap....33
8 Open questions....34
9 Summary of recommendations....35

1 INTRODUCTION

This documents existing solutions for geo-related features which could be integrated into Apertis for providing geolocation, geofencing, geocoding and navigation routing support for application bundles.

As of version 0.2.0, the recommended solutions for most of the geo-requirements for Apertis are already implemented as open source libraries which can be integrated into Apertis. Some of them require upstream work to add smaller missing features.

New components need to be written for offline address completion and geocoding, as existing solutions for these are only available as online services.

The major considerations with all of these features are:

- Whether the feature needs to work offline or can require the vehicle to have an internet connection.
- Privacy sensitivity of the data used or made available by the feature — for example, access to geolocation or navigation routing data is privacy sensitive as it gives the user's location.
- All features must support pluggable backends to allow proprietary solutions to be used if provided by the automotive domain.

2 TERMINOLOGY AND CONCEPTS

2.1 COORDINATES

Throughout this document, *coordinates* (or *a coordinate pair*) are taken to mean a latitude and longitude describing a single point in some well-defined coordinate system (typically WGS84).

2.2 GEOLOCATION

Geolocation is the resolution of the vehicle's current location to a coordinate pair. It might not be possible to geolocate at any given time, due to unavailability of sensor input such as a GPS lock.

2.3 FORWARD GEOCODING

Forward geocoding is the lookup of the zero or one addresses which correspond to a coordinate pair.

2.4 REVERSE GEOCODING

Reverse geocoding is the parsing of an address or textual description of a location, and returning zero or more coordinates which match it.

2.5 GEOFENCING

Geofencing is a system for notifying application bundles when the vehicle enters a pre-defined 'fenced' area. For example, this can be used for notifying about jobs to do in a particular area the vehicle is passing through, or for detecting the end of a navigation route.

2.6 ROUTE PLANNING

Route planning is where a start, destination and zero or more via-points are specified by the user, and the system plans a road navigation route between them, potentially optimising for traffic conditions or route length.

2.7 ROUTE REPLANNING

Route replanning is where a route is recalculated to follow different roads, without changing the start, destination or any via-points along the way. This could happen if the driver took a wrong turn, or if traffic conditions change, for example.

2.8 ROUTE CANCELLATION

Route cancellation is when a route in progress has its destination or via-points changed or removed. This does not necessarily happen when the vehicle is stopped or the ignition

turned off, as route navigation could continue after an over-night stop, for example.

2.9 POINT OF INTEREST

A *point of interest* (*POI*) is a specific location which someone (a driver or passenger) might find interesting, such as a hotel, restaurant, fuel station or tourist attraction.

2.10 TEXT-TO-SPEECH (TTS)

Text-to-speech (*TTS*) is a user interface technology for outputting a user interface as computer generated speech.

3 USE CASES

A variety of use cases for application bundle usage of geo-features are given below. Particularly important discussion points are highlighted at the bottom of each use case.

In all of these use cases, unless otherwise specified, the functionality must work regardless of whether the vehicle has an internet connection. i.e. They must work offline.

3.1 RELOCATING THE VEHICLE

If the driver is driving in an unfamiliar area and thinks they know where they are going, then realises they are lost, they must be able to turn on geolocation and it should pinpoint the vehicle's location on a map if it's possible to attain a GPS lock or get the vehicle's location through other means.

3.2 AUTOMOTIVE BACKEND

A derivative of Apertis may wish to integrate its own geo-backend, running in the automotive domain, and providing all geo-functionality through proprietary interfaces. The system integrators may wish to use some functionality from this backend and other functionality from a different backend. They may wish to ignore some functionality from this backend (for example, if its implementation is too slow or is missing) and not expose that functionality to the app bundle APIs at all (if no other implementation is available).

3.2.1 CUSTOM AUTOMOTIVE BACKEND FUNCTIONALITY

The proprietary geo-backend in a derivative of Apertis may expose functionality beyond what is described in this design, to be used by app bundles provided by the system integrator. This functionality may be exposed directly by the backend and used by applications using specific APIs on the backend.

3.3 SDK BACKEND

Developers using the Apertis SDK to develop applications must have access to geo-functionality during development. All geo-functionality must be implemented in the SDK.

The SDK can be assumed to have internet access, so these implementations may rely on the internet for their functionality.

3.4 DIFFERENT TYPES OF VEHICLE

When calculating a navigation route (Navigating between two places), the system should support calculating routes for different types of vehicle — the system could be deployed in a car, or a motorbike, or a HGV, for example. Different roads are available for use by these different vehicles.

3.5 ROUTE CALCULATION FOR PUBLIC TRANSPORT

When calculating a navigation route, the system should be able to calculate a route

between the same start, via-points and destination using public transport, for example to provide a comparison against the car route; or to incorporate public transport schemes such as park-and-ride into its route suggestions.

3.6 AUDIO-ONLY ROUTE GUIDANCE

If the driver is navigating a route (Navigating between two places), the system should support audio turn-by-turn navigation, reading out route instructions to the driver as they are approached. This allows the driver to avoid looking at the IVI screen to see the map, allowing them to focus their eyes on the road.

Route guidance must continue even if another application takes the foreground focus on the IVI system.

3.7 VIEWING AN ADDRESS ON THE MAP

The driver is attempting to get to a location nearby, and they do not want to do a full navigation route; they have the destination address and want to see a map of the area around it. They must be able to enter the address and see it displayed on the map.

In order to see the surrounding area, the map should be a 2D top-down atlas-style view.

3.8 FINDING THE ADDRESS OF A LOCATION ON THE MAP

The driver has found somewhere on the map which they want to drive to, but they do not know its address and wish to copy that down to their phone. The system must be able to display the address of a selected building or area, if it is known to the system's map provider.

3.9 TYPE-AHEAD SEARCH AND COMPLETION FOR ADDRESSES

When entering a textual address for navigation, the system may provide completion options to the user as they type, if any potential completion addresses are known to the system's map provider. This allows the user to speed up entry of addresses, and bypass making typos on longer addresses while typing when the vehicle is moving (if the passenger is entering an address, for example).

This functionality is optional — if not implemented, or if no completion information is available for the current region, the system must not provide completion suggestions.

3.10 NAVIGATING BETWEEN TWO PLACES

The driver must be able to specify a start and destination address and have the system calculate a driving route between the two, optimising for minimal travel time (or potentially for other factors). The system should display an overview of the calculated route and provide turn-by-turn directions (Turn-by-turn navigation directions). The system must track the driver's current location on the map throughout the navigation.

In order to better match the driver's view out of the windscreen, the map displayed when

navigating should be a 3D projection to better emphasise navigational input which is coming up sooner (for example, the nearest junction or road signs).

3.11 CHANGING DESTINATION DURING NAVIGATION

If the driver changes their mind about which destination to go to en-route to their original destination, the system must support changing the destination and calculating a new route.

3.12 NAVIGATING VIA WAYPOINTS

When entering a route, the driver may want to go via a specific intermediate location before heading to their final destination. The system must allow the driver to enter zero or more via-points which the route must go between, in order. The calculated route should still be optimised for minimal travel time (or some other criterion). As the driver's plans might change en-route, the system must allow changing the set of via-points while part-way through navigation.

3.13 ESTIMATED TIME LEFT IN A NAVIGATION

While navigating, the system must provide the driver with up-to-date information about the estimated travel time and distance left on their route, plus the total elapsed travel time since starting the route. This allows the driver to work out when to take rest breaks from driving.

3.14 DEVIATING OFF A NAVIGATION ROUTE

If the driver takes a wrong turning while navigating, the system must detect this and recalculate the route to bring them back en-route to their destination, reoptimising the route for minimal travel time (or some other criterion) from their new location.

The system must support calculating a radically different route if the vehicle's new location makes the previous route infeasible.

The route recalculation should happen within a short time period (on the order of ten seconds) so that the driver gets updated routing information quickly.

3.15 CHANGING NAVIGATION ROUTE DUE TO TRAFFIC CONDITIONS

The system must support recalculating and potentially changing a navigation route if traffic conditions change and make the original route take significantly longer than an alternative route.

There must be some form of hysteresis on these recalculations so that two routes which have very similar estimated travel times, but which keep alternating as the fastest, do not continually replace each other as the suggested route.

The system may ask or inform the driver about route recalculations, as the driver may be able to assess and predict the traffic conditions better than the system.

3.16 TURN-BY-TURN NAVIGATION DIRECTIONS

While navigating, the system must provide turn-by-turn navigation directions to the driver if desired by them. This includes directions about signage, upcoming turns, speed limits, speed cameras, etc.

3.17 NAVIGATION ROAD INFORMATION

While navigating, the system must provide information to the driver about the roads the vehicle is travelling along or heading towards and going to turn on to in future, such as the road name and number, and major towns or cities the road leads to. This allows the driver to match up the turn-by-turn navigation directions (Turn-by-turn navigation directions) with on-road signage.

3.18 RESUMING A NAVIGATION ROUTE

If the driver takes a rest break during a navigation route, and turns the vehicle off, the system must give the driver the option to resume the navigation route when the vehicle is turned on again. The route must be recalculated from the vehicle's current location to ensure the resumed route is still optimal for current traffic conditions.

3.19 CANCELLING A NAVIGATION ROUTE

As well as resuming the route in Resuming a navigation route, the vehicle must support cancelling the navigation when the vehicle is turned on again, at which point all navigation and turn-by-turn directions stop.

3.20 TASKS NEARBY

A to-do list application may allow the user to associate a location with a to-do list item, and should display a notification if the vehicle is driven near that location, reminding the driver that they should pop by and do the task. Once the task is completed or removed, the geo-fenced notification should be removed.

3.21 TURNING ON HOUSE LIGHTS

A 'smart home' application may be able to control the user's house lights over the internet. If the vehicle is heading towards the user's house, the app should be able to detect this and set turn the lights on over the internet to greet the user when they get home.

3.22 TRAVELLING ABROAD

The driver must be able to take the vehicle abroad and it should provide locale-sensitive navigation information, such as speed limits in the local units, and road descriptions which match the local signage conventions.

3.23 TEMPORARY LOSS OF GPS SIGNAL

When going through a tunnel, for example, the vehicle may lose sight of GPS satellites and no longer have a GPS fix. The system must continue to provide an estimated vehicle location to apps, with suitably increasing error bounds. This location may be calculated by dead reckoning from the last known GPS fix and vehicle trajectory, for example.

3.24 LOOK UP THE ADDRESS OF A CONTACT

The driver wants to navigate to a friend's house, but cannot remember their address. They have the address in their address book. The driver should be able to open the navigation app, enter the name of their friend, and the app will look up the friend's address for navigation.

3.25 NAVIGATION-DRIVEN REFINEMENT OF GEOLOCATION

The location reported by the geolocation APIs may be refined by input from the navigation system, such as snapping the location to the nearest road, or supplementing it with dead-reckoning data based on the vehicle's velocity history.

3.26 APPLICATION-DRIVEN REFINEMENT OF GEOLOCATION

If the user installs an application bundle from a new restaurant chain ('Hamburger Co', who are new enough that their restaurants are not in commercial mapping datasets yet), and wants to search for such a restaurant in a particular place (London), they may enter 'Hamburger Co, London'. The application bundle should expose its restaurant locations as a general point of interest stream[1], and the geocoding system should query that in addition to its other sources.

3.26.1 EXCESSIVE RESULTS FROM APPLICATION-DRIVEN REFINEMENT OF GEOLOCATION

A badly written application bundle which exposes a general point of interest stream might return an excessive number of results for a query — either results which are not relevant to the current geographic area, or too many results to reasonably display on the current map.

3.27 MALWARE APPLICATION BUNDLE

A malicious developer may produce a malware application bundle which, when installed, tracks the user's vehicle to work out opportune times to steal it. This should not be possible.

1 https://wiki.apertis.org/Points_of_interest#General_POI_providers

4 NON-USE-CASES

4.1 POI DATA

Handling of points of interest is covered by the Points of interest design[2]. This includes searching for points of interest nearby, displaying points of interest while driving past them, and looking up information about points of interest.

4.2 BEACONS

The iOS Location and Maps API supports advertising a device's location[3] using a low-power beacon, such as Bluetooth. This is not a design goal for the Apertis geo-APIs.

4.3 LOADING MAP TILES FROM A BACKEND

There is no use case for implementing 2D map rendering via backends and (for example) loading map tiles from a backend in the automotive domain. 2D map rendering can be done entirely in the IVI domain using a single libchamplain backend (tile source).

This may change in future iterations of this document.

2 https://wiki.apertis.org/Points_of_interest

3 https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/RegionMonitoring/RegionMonitoring.html#//apple_ref/doc/uid/TP40009497-CH9-SW1

5 REQUIREMENTS

5.1 GEOLOCATION API

Geolocation using GPS must be supported. Uncertainty bounds must be provided for the returned location, including the time at which the location was measured (in order to support cached locations). The API must gracefully handle failure to geolocate the vehicle (for example, if no GPS satellites are available).

Locations must be provided with speed, altitude and bearing information if known.

See Relocating the vehicle.

5.2 GEOLOCATION SERVICE SUPPORTS SIGNALS

Application bundles must be able to register to receive a signal whenever the vehicle's location changes significantly. The bundle should be able to specify a maximum time between updates and a maximum distance between updates, either of which may be zero. The bundle should also be able to specify a *minimum* time between updates, in order to prevent being overwhelmed by updates.

See Navigating between two places, Estimated time left in a navigation, Deviating off a navigation route, Turn-by-turn navigation directions.

5.3 GEOLOCATION IMPLEMENTS CACHING

If an up-to-date location is not known, the geolocation API may return a cached location with an appropriate time of measurement.

See Relocating the vehicle, Tasks nearby.

5.4 GEOLOCATION SUPPORTS BACKENDS

The geolocation implementation must support multiple backend implementations, with the selection of backend or backends to be used in a particular distribution of Apertis being an integration time decision.

See Automotive backend.

5.5 NAVIGATION ROUTING API

Planning a route between two locations must be supported (see Navigating between two places), including support for via-points (see Navigating via waypoints). There must be a way to update the route as the journey progresses (see Changing navigation route due to traffic conditions).

It is assumed that other use cases requiring route recalculation can be handled similarly as the initial calculation of a route (see Resuming a navigation route, Deviating off a navigation route, Changing destination during navigation).

Navigation routing must work when the vehicle has no internet connection.

See Automotive backend.

5.6 NAVIGATION ROUTING SUPPORTS BACKENDS

The navigation routing implementation must support multiple backend implementations, with the selection of backend or backends to be used in a particular distribution of Apertis being an integration time decision.

See Automotive backend.

5.7 TYPE-AHEAD SEARCH AND ADDRESS COMPLETION SUPPORTS BACKENDS

The address completion implementation must support multiple backend implementations, with the selection of backend or backends to be used in a particular distribution of Apertis being an integration time decision.

See Automotive backend.

5.8 GEOCODING SUPPORTS BACKENDS

The geocoding implementation must support multiple backend implementations, with the selection of backend or backends to be used in a particular distribution of Apertis being an integration time decision.

See Automotive backend.

5.9 SDK HAS DEFAULT IMPLEMENTATIONS FOR ALL BACKENDS

A free software, default implementation of all geo-functionality must be provided in the SDK, for use by developers. It may rely on an internet connection for its functionality.

The SDK implementation must support all functionality of the geo-APIs in order to allow app developers to test all functionality used by their applications.

See SDK backend.

5.10 SDK APIS DO NOT VARY WITH BACKEND

App bundles must not have to be modified in order to switch backends: the choice of backend should not affect implementation of the APIs exposed in the SDK to app bundles.

See Automotive backend.

5.11 NAVIGATION ROUTING SUPPORTS VEHICLE TYPES

The routing library must support calculating routes for different vehicle types, including (for example) cars, motorbikes, HGVs, pedestrians and various types of public transport. It must support defining the set of vehicle types to use for planning a particular route, so that routes which use a combination of vehicle types can be calculated (for example,

driving to a park-and-ride station, taking a bus for a short distance, then walking to the destination).

See Different types of vehicle, Route calculation for public transport.

5.12 NAVIGATION ROUTING SUPPORTS ALTERNATIVE ROUTES

The routing library must support providing one or more alternative routes for a given navigation, to allow the user to choose their preferred one.

See Changing navigation route due to traffic conditions.

5.13 TURN-BY-TURN INSTRUCTIONS ARE SUITABLE FOR TTS

Turn-by-turn navigation instructions generated by the routing library must be suitable to be read out by a TTS system. If necessary, this might mean including metadata in the instructions to assist with pronunciation or emphasis of important words.

See Audio-only route guidance.

5.14 TTS ROUTE GUIDANCE MUST BE IMPLEMENTED AS A SERVICE

The route guidance and text to speech service must be a service rather than an application, so that route guidance can continue to function In the background if a second application is started in the foreground.

See Audio-only route guidance.

5.15 2D MAP RENDERING API

Map display has the following requirements:

- Rendering the map (see Relocating the vehicle).
- Rendering the vehicle's current location (see Relocating the vehicle).
- Rendering points of interest, including start and destination points for navigation.
- Rendering a path or route.
- Rendering a polygon or region highlight.
- The map display must support loading client side map tiles, or server-provided ones.

See Viewing an address on the map.

Open question: Should map rendering be done client-side (vector maps) or pre-computed (raster maps)?

5.16 REVERSE GEOCODING API

Reverse geocoding must be supported for address entry when creating a navigation route,

converting an address into zero or more coordinates (see Viewing an address on the map). Limiting the search results to coordinates in a radius around a given reference coordinate pair must be supported.

Reverse geocoding must work when the vehicle has no internet connection.

See Viewing an address on the map, Look up the address of a contact.

5.17 FORWARD GEOCODING API

Forward geocoding must be supported for querying addresses at selected coordinates on the map when creating a navigation route (see Finding the address of a location on the map). Limiting the search results to a given radius around the coordinates, and a certain number of results, should be supported.

Forward geocoding must work when the vehicle has no internet connection.

See Finding the address of a location on the map.

5.18 TYPE-AHEAD SEARCH AND ADDRESS COMPLETION API

Suggesting and ranking potential completions to a partially entered address must be supported by the system, with latency suitable for use in a type-ahead completion system.

Address completion must work when the vehicle has no internet connection.

See Type-ahead search and completion for addresses.

5.19 NAVIGATION ROUTING SUPPORTS WAYPOINTS

The routing library must support the use of zero or more via-points in route calculations, and must support changing the set of via-points and recalculating the route after the initial calculation.

See Navigating between two places.

5.20 NAVIGATION ROUTING SUPPORTS DIFFERENT OPTIMISATIONS

The routing library must support choosing different ways to optimise route calculation; for example, for shortest distance or shortest time.

See Navigating between two places.

Open question: Does navigation routing need to take current traffic conditions into account?

5.21 NAVIGATION ROUTING PROVIDES TURN-BY-TURN INSTRUCTIONS

The routing library must be able to return information about waypoints along the way where navigation actions need to be taken (such as turning off a road) (see Turn-by-turn navigation directions).

See Navigating between two places.

5.22 3D MAP RENDERING API

For rendering the map and navigation instructions when navigating, a 3D map display should be provided, which uses a 'bird's view' projection above the vehicle to give prominence to navigational cues which are coming up sooner.

See Navigating between two places.

5.23 NAVIGATION ROUTING PROVIDES STATISTICS

The routing library must be able to return estimated data, such as the time of arrival, distance between start and destination locations (along the route), and distance between current location and destination (along the route).

See Estimated time left in a navigation.

5.24 NAVIGATION ROUTING PROVIDES ROAD INFORMATION

The routing library must provide information about roads which are relevant to the current set of turn-by-turn navigation instructions, such as road names, road numbers, and major cities which the roads head towards.

See Navigation road information.

5.25 NAVIGATION ROUTING CAN WAKE UP APPLICATIONS

Application bundles must be able to request signals when:

- reaching a navigation waypoint or destination (this is a form of Error: Reference source not found, Navigating via waypoints);
- the route is replanned (see Resuming a navigation route);
- the route is cancelled (see Cancelling a navigation route).

It must be possible for these signals to be delivered even if the bundle is no longer running.

See Deviating off a navigation route, Turning on house lights.

5.26 GEOFENCING API

Application bundles must be able to define arbitrary regions – either arbitrary polygons, or points with radii – and request a signal when entering, exiting, or dwelling in a region. The vehicle is dwelling in a region if it has been in there for a specified amount of time without exiting.

See Cancelling a navigation route, Tasks nearby.

5.27 GEOFENCING SERVICE CAN WAKE UP APPLICATIONS

It must be possible for geofencing signals to be delivered even if the application bundle which registered to receive them is not currently running.

See Cancelling a navigation route, Tasks nearby.

5.28 NAVIGATION ROUTING MUST BE LOCALE-AWARE

Routing information, such as speed limits, and the side of the road which is driven on, must be localisable to the country the vehicle is currently in (see Travelling abroad). For example, the system must report speed limits in the same units as are used on road signage.

See Travelling abroad.

5.29 GEOCODING API MUST BE LOCALE-AWARE

The geocoding API must support returning results or taking input, such as addresses, in a localised form. The localisation must be configurable so that, for example, the user's home locale could be used, or the locale of the country the vehicle is currently in.

See Travelling abroad.

5.30 GEOLOCATION PROVIDES ERROR BOUNDS

The geolocation API must provide an error bound for each location measurement it returns, so calling code knows how accurate that data is likely to be.

See Temporary loss of GPS signal.

5.31 GEOLOCATION IMPLEMENTS DEAD-RECKONING

The geolocation API must implement dead reckoning based on the vehicle's previous velocity, to allow a location to be returned even if GPS signal is lost. This must update the error bounds appropriately (Geolocation provides error bounds).

See Temporary loss of GPS signal.

5.32 GEOLOCATION USES NAVIGATION DATA IF AVAILABLE

If such data is available, the geolocation API must use navigation data to improve the accuracy of the location it reports, for example by snapping the GPS location to the nearest road on the map.

See Navigation-driven refinement of geolocation.

5.33 GEOCODING USES GENERAL POINTS OF INTEREST STREAMS

The geocoding API must be able to query general points of interest streams exposed by applications to it, and return such points of interest in its result set for forward or reverse

geocoding queries.

See Application-driven refinement of geolocation.

5.34 LOCATION INFORMATION REQUIRES PERMISSIONS TO ACCESS

There are privacy concerns with allowing bundles access to location data. The system must be able to restrict access to any data which identifies the vehicle's current, past or planned location, unless the user has explicitly granted a bundle access to it. The system may differentiate access into coarse-grained and fine-grained, for example allowing application bundles to request access to location data at the resolution of a city block, or at the resolution of tens of centimetres. Note that fine-grained data access must be allowed for geofencing support, as that essentially allows bundles to evaluate the vehicle's current location against arbitrary location queries.

Application bundles asking for fine-grained location data must be subjected to closer review when submitted to the Apertis application store.

See Malware application bundle.

Open question: What review checks should be performed on application bundles which request permissions for location data?

5.35 RATE LIMITING OF GENERAL POINT OF INTEREST STREAMS

When handling general point of interest streams generated by applications, the system must prevent denial of service attacks from the applications by limiting the number of points of interest they can feed to the geolocation and other services, both in the rate at which they are transferred, and the number present in the system at any time.

See Excessive results from application-driven refinement of geolocation.

6 EXISTING GEO SYSTEMS

This chapter describes the approaches taken by various existing systems for exposing sensor information to application bundles, because it might be useful input for Apertis' decision making. Where available, it also provides some details of the implementations of features that seem particularly interesting or relevant.

6.1 W3C GEOLOCATION API

The W3C Geolocation API[4] is a JavaScript API for exposing the user's location to web apps. The API allows apps to query the current location, and to register for notifications of position changes. Information about the age of location data (to allow for cached locations) is returned. Information is also provided about the location's accuracy, heading and speed.

6.2 ANDROID PLATFORM LOCATION API

The Android platform location API[5] is a low-level API for performing geolocation based on GPS or visible Wi-Fi and cellular networks, and does not provide geofencing or geocoding features. It allows geolocation and cached geolocation queries, as well as notifications of changes in location. Its design is highly biased towards making apps energy efficient so as to maintain mobile battery life.

6.3 GOOGLE LOCATION SERVICES API FOR ANDROID

The Google Location Services API for Android[6] is a more fully featured API than the platform location API, supporting geocoding and geofencing in addition to geolocation. It requires the device to be connected to the internet to access Google Play services. It hides the complexity of calculating and tracking the device's location much more than the platform location API.

It allows apps to specify upper and lower bounds on the frequency at which they want to receive location updates. The location service then calculates updates at the maximum of the frequencies requested by all apps, and emits notifications at the minimum of this and the app's requested upper frequency bound.

It also defines the permissions required for accessing location data more stringently, allowing coarse- and fine-grained access.

6.4 IOS LOCATION AND MAPS API

The iOS Location Services and Maps API[7] is available on both iOS and OS X. It supports many features: geolocation, geofencing, forward and reverse geocoding, navigation

4 http://www.w3.org/TR/geolocation-API/
5 http://developer.android.com/guide/topics/location/strategies.html
6 http://developer.android.com/training/location/index.html
7 https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/Introduction/Introduction.html#//apple_ref/doc/uid/TP40009497-CH1-SW1

routing, and local search.

For geolocation, it supports querying the location and location change notifications, including notifications to apps which are running in the background.

Its geofencing support is for points and radii, and supports entry and exit notifications but not dwell notifications. Instead, it supports hysteresis based on distance from the region boundary.

Geocoding uses a network service; both forward and reverse geocoding are supported.

The MapKit API[8] provides an embeddable map renderer and widget, including annotation and overlay support.

iOS (but not OS X) supports using arbitrary apps as routing providers for rendering turn-by-turn navigation instructions[9]. An app which supports this must declare which geographic regions it supports routing within (for example, a subway navigation app for New York would declare that region only), and must accept routing requests as a URI handler. The URIs specify the start and destination points of the navigation request.

It also supports navigation routing using a system provider, which requires a network connection. Calculated routes include metadata such as distance, expected travel time, localised advisory notices; and the set of steps for the navigation. It supports returning multiple route options for a given navigation.

The local search API[10] differs from the geocoding API in that it supports *types* of locations, such as 'coffee' or 'fuel'. As with geocoding, the local search API requires a network connection.

6.5 GNOME APIS

GNOME uses several libraries to provide different geo-features. It does not have a library for navigation routing.

6.5.1 GEOCLUE

GeoClue[11] is a geolocation service which supports multiple input backends, such as GPS, cellular network location and Wi-Fi based geolocation.

Wi-Fi location uses the Mozilla Location Service[12] and requires network connectivity.

It supports geolocation notifications[13] with a minimum distance between notifications, but no time-based limiting. It does not support geofencing, but the developers are

8 https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/MapKit/MapKit.html#//apple_ref/doc/uid/TP40009497-CH3-SW1
9 https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/ProvidingDirections/ProvidingDirections.html#//apple_ref/doc/uid/TP40009497-CH8-SW5
10 https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/LocationAwarenessPG/EnablingSearch/EnablingSearch.html#//apple_ref/doc/uid/TP40009497-CH10-SW1
11 http://freedesktop.org/wiki/Software/GeoClue/
12 https://wiki.mozilla.org/CloudServices/Location
13 http://www.freedesktop.org/software/geoclue/docs/gdbus-org.freedesktop.GeoClue2.Client.html#gdbus-signal-org-freedesktop-GeoClue2-Client.LocationUpdated

interested in implementing it.

GeoClue's security model allows permissions to be applied to individual apps, and location accuracy to be restricted on a per-app basis. However, this model is currently incomplete and does not query the system's trusted computing base (TCB) (see the Security design for definitions of the TCB and trust).

6.5.2 GEOCODE-GLIB

Geocode-glib[14] is a library for forward and reverse geocoding. It uses the Nominatim API, and is currently hard-coded to query nominatim.gnome.org[15][16]. It requires network access to perform geocoding. The Nominatim API[17] does not require an API key (though it does require a contact e-mail address), but it is highly recommended that anyone using it commercially runs their own Nominatim server.

geocode-glib is tied to a single Nominatim server, and does not support multiple backends.

6.5.3 LIBCHAMPLAIN

libchamplain[18] is a map rendering library, providing a map widget which supports annotations and overlays. It supports loading or rendering map tiles from multiple sources.

6.5.4 NAVIT

NavIt[19] is a 3D turn-by-turn navigation system designed for cars. It provides a GTK+ or SDL interface, audio output using espeak, GPS input using gpsd, and multiple map rendering backends. It seems to expose some of its functionality as a shared library (libnavit), but it is unclear to what extent it could be re-used as a component in a larger system, without restructuring work.

6.6 NAVIGATION ROUTING SYSTEMS

Three alternative routing systems are described briefly below; a full analysis based on running many trial start and destination routing problems against them is yet to be done.

6.6.1 GRAPHHOPPER

GraphHopper[20] is a routing system written in Java, which is available as a server or as a library for offline use. It uses OpenStreetMap data, and is licenced under Apache License 2.0.

14 https://developer.gnome.org/geocode-glib/stable/
15 https://bugzilla.gnome.org/show_bug.cgi?id=756311
16 https://bugzilla.gnome.org/show_bug.cgi?id=756313
17 http://wiki.openstreetmap.org/wiki/Nominatim
18 https://wiki.gnome.org/Projects/libchamplain
19 http://www.navit-project.org/
20 https://graphhopper.com/

6.6.2 OSRM

OSRM[21] is a BSD-licensed C++ routing system, which can be used as a server or as a library for offline use. It uses OpenStreetMap data.

6.6.3 YOURS

YOURS[22] is an online routing system which provides a web API for routing using OpenStreetMap data.

6.7 NAVSERVER

NavServer is a proprietary middleware navigation solution, which accesses a core navigation system over D-Bus.

It was not possible to view the documentation or APIs it exposes at the time this design was written (version 0.2.1).

Open question: What functionality and APIs does NavServer provide?

21 http://project-osrm.org/
22 http://wiki.openstreetmap.org/wiki/YOURS

7 APPROACH

Based on the above research (section 6) and requirements (section 5), we recommend the following approach for integrating geo-features into Apertis. The overall summary is to use existing Freedesktop.org and GNOME components for all geo-features, adding features to them where necessary, and adding support for multiple backends to support implementations in the automotive domain.

7.1 BACKENDS

Each of the geo-APIs described in the following sections will support multiple backends. These backends must be chosen at compile time. They do not need to be switcheable at runtime, and app bundles must not be able to choose which backend is being used for a particular geo-function — that is chosen by the system integrator, who knows better than the application developer about which geo-backends are available and what their strengths and weaknesses are.

If there are particular situations where it is felt that the application developer knows better than the system integrator about the backend to use, that signals a use case which has not been considered, and might be best handled by a revision of this design and potentially introducing a new SDK API to expose the backend functionality desired by the application developer.

Runtime switching of backends would also lead to problems of transferring state between the old backend and the new one, such as route planning information or GPS location history.

Backends may be implemented in the IVI domain (for example, the default backends in the SDK must be implemented in the IVI domain, as the SDK has no other domains), or in the automotive domain. If a backend is implemented in the automotive domain, its functionality must be exposed as a proxy service in the IVI domain, which implements the SDK API. The IPC mechanism for communicating between this proxy service and the backend itself is not defined in this document. This IPC interface serves as a security boundary for the backend.

7.2 2D MAP DISPLAY

libchamplain[23] should be used for top-down map display. It supports map rendering, adding markers for the vehicle's current location, points of interest (with explanatory labels), and start and destination points. Paths, routes, polygons and region highlights can be rendered as using Clutter API on custom map layers.

libchamplain supports pre-rendered tiles from online (`ChamplainNetworkTileSource`) or offline (`ChamplainFileTileSource`) sources. It supports rendering tiles locally using libmemphis, if compiled with that support enabled.

On an integrated system, map data will be available offline on the vehicle's file system. On the Apertis SDK, an internet connection is always assumed to be available, so map tiles

23 https://wiki.gnome.org/Projects/libchamplain

may be used from online sources. libchamplain supports both.

7.3 3D MAP DISPLAY

Our initial suggestion is to use NavIt[24] for 3D map display and navigation rendering. However, we are currently unsure of the extent to which it can be used as a library, and the extent to which it could be integrated with the Apertis navigation routing API. Similarly, since we have no details on the API exposed by proprietary solutions, such as NavServer (see section 6.7), which would potentially become backends for it, we cannot yet recommend an API for 3D map display.

There are two possibilities: libnavit is suitable for use as the SDK 3D map display API, and we can commit to using it with multiple backends (which might require work to allow new backends like NavServer to be integrated with it); or libnavit is not suitable, and an abstraction API needs to be designed to wrap NavIt and NavServer, with this abstraction API becoming the supported SDK 3D map display API.

7.4 GEOLOCATION

GeoClue[25] should be used for geolocation. It supports geolocation using GPS, 3G and Wi-Fi. It supports accuracy bounds for locations[26], but does not pair that with information about the time of measurement. That would need to be added as a new feature. Cached locations are not supported; that would also need to be added. Similarly, dead-reckoning of the location based on previous velocity is not supported for when GPS signal is lost. That would need to be added as a feature.

Speed, altitude and bearing information are supported[27].

Multiple backends are supported, so closed source as well as open source backends could be used.

If the navigation backend implements a snap-to-road feature, it should be used as a further source of input to GeoClue for refining the location.

7.5 GEOLOCATION SIGNALS

GeoClue emits a `LocationUpdated` signal[28] whenever the vehicle's location changes more than the `DistanceThreshold`[29]. GeoClue currently does not support rate limiting emission of the `LocationUpdated` signal for minimum and maximum times between updates. That would need to be added to it.

24 http://www.navit-project.org/
25 http://freedesktop.org/wiki/Software/GeoClue/
26 http://www.freedesktop.org/software/geoclue/docs/gdbus-org.freedesktop.GeoClue2.Location.html#gdbus-property-org-freedesktop-GeoClue2-Location.Accuracy
27 http://www.freedesktop.org/software/geoclue/docs/gdbus-org.freedesktop.GeoClue2.Location.html
28 http://www.freedesktop.org/software/geoclue/docs/gdbus-org.freedesktop.GeoClue2.Client.html#gdbus-signal-org-freedesktop-GeoClue2-Client.LocationUpdated
29 http://www.freedesktop.org/software/geoclue/docs/gdbus-org.freedesktop.GeoClue2.Client.html#gdbus-property-org-freedesktop-GeoClue2-Client.DistanceThreshold

7.6 NAVIGATION ROUTING

Navigation routing could either be implemented on the system as a library, in which case it would be instantiated separately for each app which uses it; or it could be instantiated as a single system service. This would save memory, at the cost of requiring all navigation routing data (the parameters to and return value from routing queries[30]) to be sent over D-Bus. As these values are typically sent over a HTTP interface for online routing queries anyway, this is perfectly feasible.

We suggest implementing the new system service to calculate navigation routes and track the vehicle's progress on them. By using a system service, routing, turn-by-turn navigation and TTS output of instructions can continue while another application is in the foreground on the IVI system.

The service should support different backends, using only one of them at once. This allows for routing software from the automotive domain to be exposed to applications via the routing API.

For supporting offline routing, we recommend OSRM as the default backend, pending trialling it against some example routing queries to check whether it produces optimal routes.

OSRM supports routing between two locations, with via-points[31]. Updating the route as the journey progresses can be done by re-running the routing query, starting from the vehicle's current location.

OSRM returns a `route_summary` with estimated data about the trip distance and time. It returns turn-by-turn directions; and alternative routes if requested.

We recommend the D-Bus API of the routing service mirrors that provided by OSRM, as it has a sensible API design already.

We suggest that this API is kept separate from the 3D map API (see section 7.3), so that routing can be performed without necessarily rendering it on screen. However, as this is a common use case, the two APIs should integrate easily as well (for example, having the same format for turn-by-turn instructions).

7.7 LOCALISATION OF GEOCODING

OpenStreetMap can provide localised forms of place names[32], although the necessary data may not be in the database for all places. The convention is for place names to be stored in the local dialect by default, so this would only cause a problem for translation to the driver's language. i.e. By default, OpenStreetMap data should match on-road signage.

Nominatim supports exporting localised place names from OpenStreetMap, but geocode-glib does not currently expose that data in its query results. It would need to be modified to support this.

30 https://github.com/Project-OSRM/osrm-backend/wiki/Server-api#user-content-service-viaroute
31 https://github.com/Project-OSRM/osrm-backend/wiki/Server-api#user-content-service-viaroute
32 http://wiki.openstreetmap.org/wiki/Map_internationalization

7.8 LOCALISATION OF ROUTING INFORMATION

Routing information is present in OpenStreetMap, but default values (such as which side of the road to drive on by default, or speed limits for certain types of road) are not present; only the exceptions are present. A database of world road system defaults is needed, and might be provided by the navigation routing library.

OSRM does not accept any localisation parameters, so we must assume it returns the default OpenStreetMap names for places, which should match those on road signage.

It returns turn-by-turn navigation instructions[33] as structured data, which the routing service can then format as required, potentially into multiple localised textual forms plus other forms for TTS output.

7.9 FORWARD AND REVERSE GEOCODING

We recommend that geocode-glib[34] is used as the SDK geocoding API. geocode-glib is currently hard-coded to use GNOME's Nominatim service; it would need to be modified to support multiple backends, such as an Apertis-specific Nominatim server[35], or a geocoding service from the automotive backend. It should support querying multiple backends in parallel so that general point of interest streams can be queried.

The geocode-glib API supports forward and reverse geocoding, but does not support limiting search results to a radius around given coordinates. This feature would need to be added to geocode-glib, but not necessarily to the backends themselves.

On an integrated system, geocoding data will be available offline on the vehicle's file system. This can be used by a custom backend for geocode-glib. On the Apertis SDK, an internet connection is always assumed to be available, so geocoding may be performed using an online Nominatim backend. In both cases, geocode-glib would form the SDK API used by apps.

General point of interest streams could be fed into geocode-glib via another new backend, queried in parallel with other backends.

7.10 ADDRESS COMPLETION

Nominatim does not provide address completion services, but it is possible to implement them using a filtered version of the OpenStreetMap database data. An example is available as Photon[36].

As address completion must be available when offline, the implementation should be a local C library which auto-completes from locally downloaded OpenStreetMap data. This would need to be written from scratch, including a system for generating the local data files from OpenStreetMap exports.

The system must support loading address completion data from multiple backends, one

33 https://github.com/Project-OSRM/osrm-backend/wiki/Server-api#user-content-example-2
34 https://developer.gnome.org/geocode-glib/stable/
35 http://wiki.openstreetmap.org/wiki/Nominatim
36 http://photon.komoot.de/

of which would be the local OpenStreetMap data. Other backends may be proprietary, online or offline.

7.11 ROUTE ALERTS

Signals for route replanning and cancellation can be emitted as D-Bus signals. Apps which wish to receive them would have to register as separate clients of the routing service, as in the GeoClue API[37]. This allows different signals to be emitted to different apps depending on their permissions.

Delivery of signals to bundles which are not currently running may cause activation of that app[38].

7.12 GEOFENCING

We recommend that GeoClue is modified upstream to implement geofencing of arbitrary regions. Signals on entering or exiting a geofenced area should be emitted as a D-Bus signal which the app bundle subscribes to. Delivery of signals to bundles which are not currently running may cause activation of that app[39].

We have informally discussed the possibility of adding geofencing with the GeoClue developers, and they are in favour of the idea.

7.13 LOCATION SECURITY

libchamplain is a rendering library, and does not give access to sensitive information.

geocode-glib is a thin interface to a web service, and does not give access to sensitive information. All web service requests must be secured with best web security practices, such as correct use of HTTPS, and sending a minimum of identifiable information to the web service.

GeoClue provides access to sensitive information about the vehicle's location. It currently allows limiting the accuracy provided to apps as specified by the user[40]; this could be extended to implement a policy determined by the capabilities requested in the app's manifest.

Similarly for GeoClue's geofencing feature, when it is it added — clients have separated access to its D-Bus API to allow them to be signalled at different accuracies and rates. This applies to navigation routing as well, as it may provide feedback to applications about progress along a route, which exposes information about the vehicle's location.

Application bundles asking for fine-grained location data should be subjected to closer review when submitted to the Apertis application store.

37 http://www.freedesktop.org/software/geoclue/docs/gdbus-org.freedesktop.GeoClue2.Client.html

38 This is intended to be provided by a system service for activation of applications based on subscribed signals; the design is tracked in https://phabricator.apertis.org/T605.

39 This is intended to be provided by a system service for activation of applications based on subscribed signals; the design is tracked in https://phabricator.apertis.org/T605.

40 http://www.freedesktop.org/software/geoclue/docs/gdbus-org.freedesktop.GeoClue2.Client.html#gdbus-property-org-freedesktop-GeoClue2-Client.RequestedAccuracyLevel

7.14 SYSTEMIC SECURITY

As the geo-features can source information from application bundles, they form part of the security boundary around application bundles.

In order to avoid denial of service attacks from an application bundle which emits too much data as, for example, a general point of interest stream, the system should rate limit such streams in time (number of POIs per unit time) and space (number of POIs per map area).

Location updates emitted by GeoClue must be rate limited between the minimum and maximum distance and time limits set by each client. These limits must be checked to ensure that a client is not requesting updates too frequently.

For the components in the system which provide access to sensitive information (the vehicle's location), a security boundary needs to be defined between them and application bundles. Geolocation, navigation routing and geofencing are the sensitive APIs — these are all implemented as services, so the D-Bus APIs for those services form the security boundary.

7.15 REQUIREMENTS

This design fulfils the following requirements:

- 5.1: Geolocation API — use GeoClue
- 5.2: Geolocation service supports signals — use GeoClue; augment its signals
- 5.3: Geolocation implements caching — to be added to GeoClue
- 5.4: Geolocation supports backends — GeoClue supports backends
- 5.5: Navigation routing API — new routing service exposing an API similar to OSRM's
- 5.6: Navigation routing supports backends — routing service will support backends
- 5.7: Type-ahead search and address completion supports backends — to be implemented as a new address completion library
- 5.8: Geocoding supports backends — to be added to geocode-glib
- 5.9: SDK has default implementations for all backends — Gypsy for geolocation; custom online Nominatim server for geocoding; online OpenStreetMap for 2D maps; libnavit for 3D maps, **subject to further evaluation**; OSRM for navigation routing; new offline OpenStreetMap database for address completion
- 5.10: SDK APIs do not vary with backend — GeoClue API for geolocation; geocode-glib for geocoding; libchamplain for 2D maps; libnavit for 3D maps, **subject to further evaluation**; routing service API for navigation routing; new library for address completion
- 5.11: Navigation routing supports vehicle types — OSRM supports multiple vehicle modes; routing API is based on OSRM

- 5.12: Navigation routing supports alternative routes — OSRM supports alternative routes; routing API is based on OSRM
- 5.13: Turn-by-turn instructions are suitable for TTS — OSRM returns routes as non-locale-dependent data structures
- 5.14: TTS route guidance must be implemented as a service — routing API is implemented as a service
- 5.15: 2D map rendering API — use libchamplain with a local or remote tile store
- 5.16: Reverse geocoding API — use geocode-glib
- 5.17: Forward geocoding API — use geocode-glib
- 5.18: Type-ahead search and address completion API — to be written as a new C library
- 5.19: Navigation routing supports waypoints — OSRM supports waypoints; routing API is based on OSRM
- 5.20: Navigation routing supports different optimisations — OSRM **does not support** this, it only supports shortest route calculations; other backends may support it
- 5.21: Navigation routing provides turn-by-turn instructions — OSRM provides turn-by-turn instructions; routing API is based on OSRM
- 5.22: 3D map rendering API — use libnavit, **subject to further evaluation**
- 5.23: Navigation routing provides statistics — OSRM provides statistics; routing API is based on OSRM
- 5.24: Navigation routing provides road information — OSRM provides road names and directions; routing API is based on OSRM
- 5.25: Navigation routing can wake up applications — to be implemented as signals from routing service to registered applications
- 5.26: Geofencing API — to be implemented as a new feature in GeoClue
- 5.27: Geofencing service can wake up applications — to be implemented as a new feature in GeoClue
- 5.28: Navigation routing must be locale-aware — OSRM exposes OpenStreetMap localised data
- 5.29: Geocoding API must be locale-aware — to be added to geocode-glib to expose existing OpenStreetMap localised data
- 5.30: Geolocation provides error bounds — GeoClue provides accuracy information, but it needs augmenting
- 5.31: Geolocation implements dead-reckoning — to be added to GeoClue
- 5.32: Geolocation uses navigation data if available — to be added as another backend to GeoClue

- 5.33: Geocoding uses general points of interest streams — to be implemented as another new backend for geocode-glib
- 5.34: Location information requires permissions to access — to be implemented as manifest permissions for application bundles
- 5.35: Rate limiting of general point of interest streams — security boundary implemented as D-Bus API boundary; rate limiting applied on signal emission and processed general point of interest streams

7.16 SUGGESTED ROADMAP

As the SDK APIs for geo-features are, for the most part, provided by FOSS components which are available already, the initial deployment of geo-features requires GeoClue, geocode-glib, libchamplain and OSRM to be packaged for the distribution, if they are not already. The initial deployment would also require a routing service and address completion library to be written.

The second phase would require modification of these packages to implement missing features and implement additional backends. This can happen once the initial packaging is complete, as the packages fulfil most of Apertis' requirements in their current state.

This second phase includes modifying the packages to be container-friendly, so that they can be used by compartmentalised apps without leaking sensitive data from one app to another. This requires further in-depth design work, but should require fairly self-contained changes.

8 OPEN QUESTIONS

1. **5.15**: Should map rendering be done client-side (vector maps) or pre-computed (raster maps)?
2. **5.20**: Does navigation routing need to take current traffic conditions into account?
3. **5.34**: What review checks should be performed on application bundles which request permissions for location data?
4. **6.7:** What functionality and APIs does NavServer provide?

9 SUMMARY OF RECOMMENDATIONS

As discussed in the above sections the recommendations are:

- Packaging and using libchamplain for 2D map display.
- Packaging and using libnavit for 3D map display, **subject to further investigation**.
- Packaging and using GeoClue for geolocation. It needs measurement times, cached location support and dead-reckoning to be added upstream.
- Adding minimum and maximum update periods for clients to upstream GeoClue, alongside the existing distance threshold API for location update signals.
- Adding support for geofencing to upstream GeoClue.
- Adding a new navigation backend to GeoClue to implement snap-to-road refinement of its location.
- Packaging and using geocode-glib for forward and reverse geocoding. It needs support for exposing localised place names to be added upstream.
- Add support for limiting results to a given area to geocode-glib.
- Adding support for multiple backends to geocode-glib.
- Implementing a general point of interest stream backend for geocode-glib using the Apertis point of interest APIs.
- Auditing geocode-glib to ensure it maintains data privacy by, for example, using TLS for all requests.
- Implementing a local library for address completion using vehicle-side OpenStreetMap data; plus a system for generating the local data files.
- Implementing a routing service with its D-Bus API based on OSRM's API, supporting route queries and alerts.
- Packaging and using OSRM for offline navigation routing, as a backend for the new routing service.
- Integrating GeoClue, the routing service and the geocoding service with the app service proxy to apply access control rules to whether apps can communicate with it to retrieve potentially sensitive location data. Only permit this if the appropriate coarse- or fine-grained location permission has been set on the application bundle’s manifest.