POSEiDON

PersOnalized Smart Environments to increase Inclusion of people with DOwn's syNdrome

Deliverable D5.6

Integrated POSEIDON technology technical documentation

Call:	FP7-ICT-2013-10
Objective:	ICT-2013.5.3 ICT for smart and personalised inclusion
Contractual delivery date:	M18
Actual delivery date:	30.04.2015
Version:	V1
Author:	Lars Thomas Boye, Tellu AS
Contributors:	Fenglin Han, Karde AS Dean Kramer, MU Silvia Rus, Fraunhofer
Reviewers:	Dean Kramer, MU Terje Grimstad, Karde AS
Dissemination level:	Public
Number of pages:	35

Contents

Executive summary

This deliverable contains technical documentation for the POSEIDON system. The introduction gives an overview of the system. The rest of the deliverable is a collection of various documents which are maintained separately in the project as part of the work on the prototypes and pilots.

Chapter 2 describes the needed equipment, setup of service accounts and devices, and installation of the software for pilot 1. Chapter 3 is the technical user manual for the mobile application. Chapter 4 is a developer manual for the POSEIDON web application, giving information about the technology used here. Chapter 5 documents the use and semantics of calendar data in this prototype system. Chapter 6 is the developer documentation for the context middleware, while chapter 7 documents the route data. See the introduction for references to other deliverables which contain documentation of the PSOEIDON system.

1 Introduction

1.1 POSEIDON prototype 2 overview

The figure shows the most prominent components and data flows at the present stage of the project – the second of four prototype iterations. There are three types of applications, for three different platforms. The stationary system is what is installed on a PC or laptop at the home of the primary user or some other location the primary user frequently visits. The main focus here is training and virtual reality. The Android device is the current mobile device, with the main design target being large phones and small tablets. This is what the primary user can always bring with them, with the focus being on providing guidance when out and about. The web application targets the third platform – the browser. It is a responsive web app which can be used on any device. It is primarily for the secondary user, to provide monitoring and personalisation services.

Outside of these three sub-systems are the cloud services we connect to. These provide shared storage for the different sub-systems, and services necessary for their functionality. SmartTracker is the server side of the POSEIDON framework. It provides a shared storage for tracking and context observations, as well as for preferences for personalisation. Any POSEIDON application can send context observations here and access the database, as long as it is authenticated as a registered user. We have produced framework components for handling the connection to SmartTracker, to make more POSEIDON applications connected to the framework.

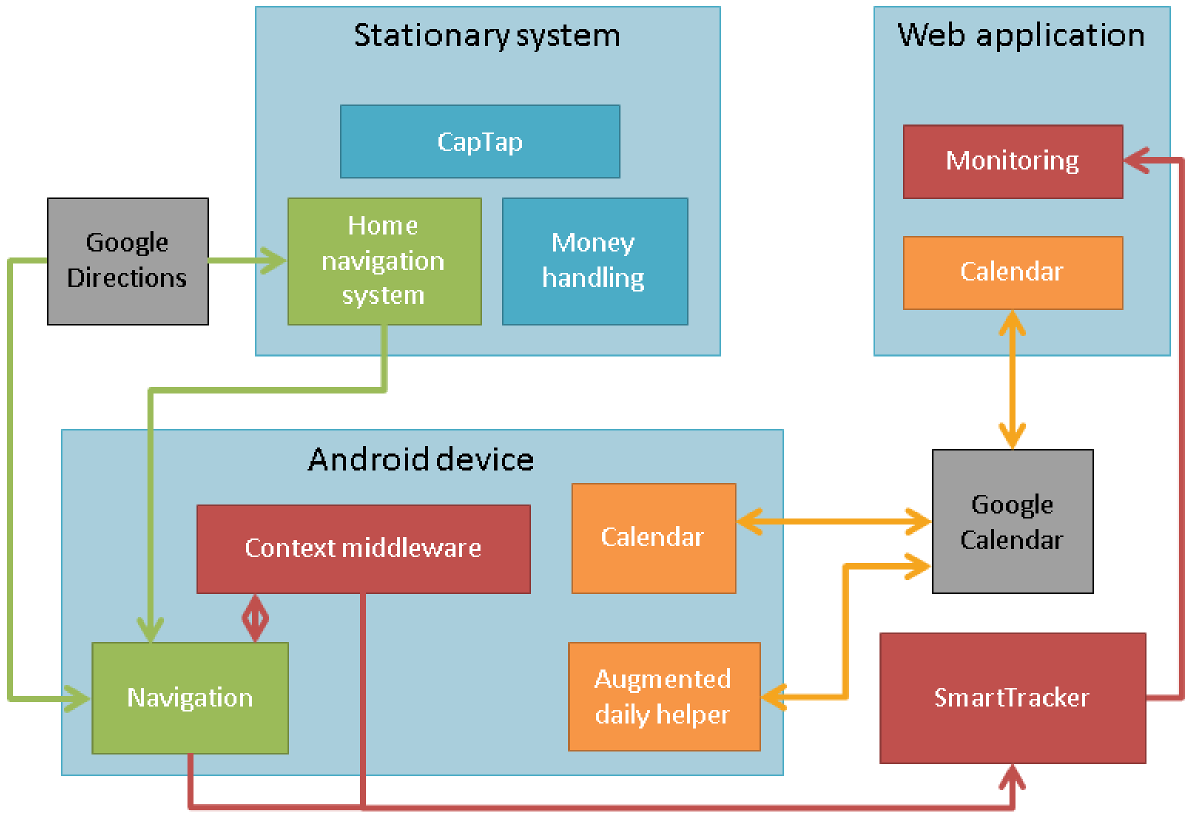


Figure 1: Overview of POSEIDON prototype 2 system

The other half of the framework is the middleware running locally on the stationary and mobile sub-systems. This is primarily for context awareness, supporting ontology and rules. For this prototype the main focus is the context middleware running on the mobile device, using the sensors of the device to monitor context. Context considered includes location, time, weather, battery capacity, communication, user preferences etc. Applications can subscribe to context updates from the middleware, and also post their own context observations to the middleware.

The colour coding in the figure shows the main aspects of this POSEIDON solution. Green is for navigation – the guidance related to space. The home system part is used by carers to create personalised routes, and by primary users to train with virtual reality. Navigation guidance is provided on the mobile device, tracking the user on a trip. The Google Directions service is used for route calculation. The basic route calculated by this service is augmented with personalised instructions and media in the home system, and the personalised routes are transferred to the mobile device.

Orange highlights the assistance related to time – the calendar-based functionality. The web application has a calendar interface, for managing events and connecting them to functionality such as navigation. The mobile application also has a calendar interface – a simpler interface better adapted to the smaller screen and primary users, but also allowing entering basic events. And the mobile application provides reminders. An augmented reality daily helper prototype has also been developed for the mobile device, to include detailed instructions including use of augmented reality for setting up and delivering calendar-based instructions. All calendar-based functionality connects to the same cloud storage of events. This is so far provided by Google, with ambitions to switch to our own storage service later.

Context awareness and tracking is indicated in red. Context and navigation tracking is sent to SmartTracker, so that it can be made available to other applications and to the monitoring part of the web application. Access to the tracked data will be restricted for privacy reasons.

On the stationary system, the other main training application in this prototype is for money handling. The *CapTap* (the interactive table) is used as an input device for both the money handling and virtual reality navigation training.

1.2 POSEIDON technical documentation

Chapter 2 is the deployment documentation for pilot 1. It describes the needed equipment, setup of service accounts and devices, and installation of the software. Chapter 3 is the technical user manual for the mobile application. For the first prototype iteration this was placed in D5.2, but it has been moved to D5.6 as this is the correct deliverable for it (there was no version of this deliverable for the first prototype). Chapter 4 is a developer manual for the POSEIDON web application, giving information about the technology used here.

Chapter 5 documents the use and semantics of calendar data in this prototype system. Chapter 6 is the developer documentation for the context middleware, while chapter 7 documents the route data.

Each chapter is a document from the technical group of the project, assembled here to create a snapshot of the documentation not covered by other deliverables, as it is on the delivery data of D5.6. Most of these documents will continue to be updated as the pilot is deployed and tested. Project personnel should refer to the separate documents in the project's internal document repository to make sure they have the latest version.

Other technical and end user documentation is found in other deliverables. See WP3 deliverables for details on the context awareness middleware layer of POSEIDON. HCI-specific documentation is found in D4.5. The PSOEIDON development framework is described in D5.1, while the data stores are documented in D5.4 and the usage of media is documented in D5.3.

2 Deployment

This is the documentation of the deployment of the system for pilot 1, describing the needed equipment, setup of service accounts and devices, and installation of the software. It is a stand-alone living document in the project, with the version of the deliverable delivery date included here.

2.1 Equipment

The following equipment is needed for the pilot trial:

- Smartphone: Huawei Ascend G7. Provided by the project. Box includes charger with USB cable and a protective cover. Applications will be pre-installed.
- SIM card/mobile subscription: Micro-SIM. The pilot primary user should get an additional SIM card for their subscription, and the project will pay for their data traffic.
- Carer needs a phone, tablet or camera for taking pictures and filming. The carer should be used to using a smartphone.
- PC/laptop with Windows 7 or higher, or Mac. The user will need to install applications for navigation training and money handling here.
- Internet and Wi-Fi at home.
- Interactive table. Provided by the project. To be connected to the PC/laptop.

2.1.1 Smartphone

We have selected the Huawei Ascend G7 for the pilot trial. It has a screen size of 5.5 inches, which we think is a very good size for the user group, being large but hopefully not crossing over into being too large. It also has a large battery, at 3000 mAh, which is important. And it is otherwise a powerful phone, with a four core processor, 2 GB memory, 16 GB storage and 4G network. It has a metal back and generally feels solid. And it is very reasonably priced, being much cheaper than equivalent phones from the more established brands. The table shows the prices in each country, including tax.

	Germany	UK	Norway
Price	258,24 €	£199.99	2995 NOK

It comes with Android version 4.4.4. As many phones are now being updated to version 5, it is likely that this phone will also get an update in the not too distant future. Huawei have their own user interface layer on top of Android, which they call EMUI. This means that things like the home screen/launcher are different from Android phones from other manufacturers, but the general principles of operation are mostly the same.

The phone comes with an optional protective cover. It’s a good idea to use it for a little added protection if the phone is dropped, but the phone is probably quite robust, so it’s not required. The cover makes the power button a bit difficult to operate. As an alternative to the power button, it is possible to turn the phone off by holding a hand over it and turning it on by double-tapping it.

If a phone is broken during the trial, we will buy a new one to replace it, but no more than once for any user and no more than two phones in total, limited by the project budget. As an incentive to the pilot families, they will get to keep the phone if they complete the trial.

2.1.2 Interactive table

The interactive table is a prototype of a new interactive device which combines the size of a multitouch table with 3D hand position recognition. It is intended to be unobtrusively built in tables. For the sake of user studies however, mobile versions have been built. These are intended to be placed on a surface in front of a monitor, like shown in Figure 2.

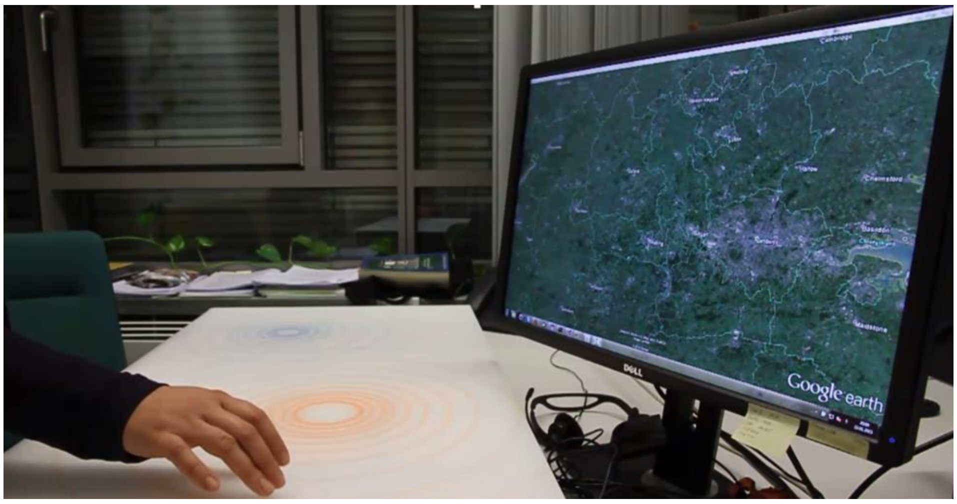


Figure 2 – mobile interactive table in front of monitor

2.2 Google account

A Google account is used to store calendar appointments. It is important that the same Google account is used on the phone and in the web application, so that they are connected and show the same content. For the pilot, the project will create a Google account for each primary pilot user. This can be done in the initial setup of the phone. The account is entered in the phone, where it is used for calendar storage and access to Google Play. For the pilot it is important that there are no other Google accounts on the phone, and that the Google account only has one calendar. Project personnel can use existing accounts or have multiple accounts on their phone for testing, but keep in mind that the mobile app doesn't distinguish between multiple calendars.

2.3 SmartTracker account

The administration interface of the SmartTracker server instance used in the POSEIDON project is available at the following URL:

http://ri.smarttracker.no

A service provider called "Poseidon SP" has been set up to manage POSEIDON accounts. Tellu manages this service provider and issues logins to relevant project partners. For the pilot we are using a single SmartTracker account to hold the user data, called "Poseidon Pilot". Each user entity, used to access data, will be restricted to one asset, representing a primary user. A SmartTracker user with administrator permissions needs to set up the SmartTracker entities for a pilot user with carer.

Firstly, create an asset to represent the primary user, and a device to represent the app. Adding a device is necessary for the application to be able to submit observations to SmartTracker. The device can be identified based on a device ID, through a different API than the one responsible for accessing accounts and data. This can be done following these steps:

- From the *Content* menu, select *Personnel*. Press *New* to create. Write a *Name*, and select "Primary user" for the *Type* property. Press *Create Person*.
- Select *Devices* from the content menu, and press *New*. Select POSEIDON app. Provide a *Name* and an *MSISDN*. MSISDN is the phone number of the primary user (the number for the SIM card used in the device). As we are so far not using the SIM number to communicate with the application, this doesn't need to be the real number.

- Go back to *Personnel* and select the new person. Click the "+" in the *Devices* field, and select the device just created.

Then we need to create a SmartTracker user for authentication. This will be used to log in to the web and app.

- From the *Administration menu*, select *Users,* and *Add user*.
- Enter *Name* in the form of an email address (i.e. @something - it doesn't have to be a valid address). Select *User* as *Role*.
- Enter then a *Password* and a *Name*. This password is temporary and must be reset before use anyway, so put something really simple.
- Tick "Do not send email verification", at least if the user *Name* isn't a valid email address, and press *Add user*.
- In the list of users, find the one just added. In the *Actions* column, click the icon with a single shield. Select the correct primary user (asset).
- Log out of SmartTracker, and try logging in with the new user. You will be asked to change the password.

If the process is successful and the asset you have access to is the right person, the account is correctly configured and can be used with the web and mobile app.

2.4 Phone setup

2.4.1 Initial startup

Insert the SIM card (micro-SIM). It goes in a small tray in the side of the phone (the lower of the two). A small tool comes with the phone – insert it in the little hole until the tray pops out.

On initial startup, there is a setup "wizard" (everything can be changed later). If the phone has already been used by someone else, first do a Factory data reset (under Backup & reset in the settings). The phone will then restart as new. The wizard goes through the following things:

- You can agree if it asks for Location consent, but there's no need to allow Google apps to access location.
- Select language.
- If Wi-Fi is available where setting up the phone, connect to it.
- Agree to terms and conditions.
- Google account: It will ask for a Google account. If a Google account hasn't already been created for the user, this is a good time to do it.
- Backup & restore: Untick Backup and Communication. The Location boxes should be ticked.
- Don't add credit card for Google services.

2.4.2 POSEIDON application

Install the POSEIDON app in Google Play[1]. Search for "poseidon tellu" and it should be the top hit for apps. Starting it, enter the SmartTracker user name and password for the user, and tick "Remember login". Once logged in, the app will stay active in the background, to occasionally check for new calendar events and other data. The POSEIDON icon will be shown in the notification bar as long as POSEIDON is active.

[1] https://play.google.com/store/apps/details?id=no.tellu.poseidon

2.4.3 POSEIDON Context Middleware

Install the POSEIDON Context Middleware app in Google Play [2]. Search for “Poseidon context”, and you should find the app “POSEIDON Context Reasoner” by “GOODIES @ Middlesex University”. Be sure to have this application installed before starting the main POSEIDON app.

2.4.4 universAAL Middleware

The universAAL middleware application needs to be installed on the device. This app is not available on the Google Play Store. Therefore, we will need to first allow apps to be installed outside of the Google Play store. This is done by firstly going to the device app **Settings.** Then scroll down to **Security.** Following this scroll down to the **Device administration** section, and enable **Unknown sources – Allow installation of non-Market apps**. Following this, the application can be installed. To do this, go to the device web browser and type in the following case sensitive web address: http://goo.gl/awQi1i . On the Google Drive webpage, click on the **Download** button, which should download the app and ask if you wishes to install it.

2.4.5 Phone settings

The phone settings are available from the launcher (a gear icon).

- Wi-Fi: Connect to the Wi-Fi in the home of the primary user.
- Display:
  - Font size: The current prototype app has been designed with the “Large” setting in mind, and this is the recommended size. The text of the POSEIDON app is in any case much larger than most apps.
  - Brightness: Increase the slider to maximum, but leave it on Automatic.
  - Sleep: You may want to increase the screen sleep time from 30 seconds to 1 minute, but it also depends on personal preferences.
  - Auto-rotate screen: Can be left on if the user is OK with it. The current prototype app will in any case always be in portrait mode.
- Sound: Phone and Notification ringtones can be changed for personal preferences.
- Power saving: The Power plan should be Normal or Smart (make sure it’s not set to Ultra).
- Location services: Make sure Mode is High accuracy.
- Notification manager: Find POSEIDON in the list, and select Allow.
- Protected apps: Find POSEIDON in the list, and turn it on (so it says Protected). This is important to make sure it is allowed to run in the background.
- Security: The Screen lock is a personal preference. If using the phone cover, the power switch isn’t easily activated by accident, so it may not be necessary to have any lock screen. The Huawei unlock (default) is a swipe unlock. A secure unlock shouldn’t be necessary. SIM card lock can also be turned off, so that it’s not necessary to enter a PIN code when turning on the phone.
- Google: Make sure a Google account has been set. Going into the account’s sync settings, make sure Calendar, Contacts and Gmail are on. The others can be disabled to save data and battery.

[2] https://play.google.com/store/apps/details?id=org.poseidon_project.context

- Motion control: As an alternative to using the power button, you can try to turn on Cover screen and Double touch. The screen can then be turned off by pressing on it with a whole hand, and turned on by tapping twice. This can be tried by the primary user.
- Date & time: Automatic date & time and time zone should work well, just check that the clock is correct. Use 24-hour format or not based on personal preference.

2.4.6 Other configuration

2.4.6.1 Google Play
Enter the Settings in the menu (the three lines in the top left). Turn off notifications. Auto-update apps should be on Wi-Fi only (the default). Any update to the POSEIDON apps put into Google Play will then be received automatically.

2.4.6.2 Calendar
Enter the phone's own calendar app. Enter Settings from the Menu. Turn off Notifications. This is important, as the POSEIDON app should be the only calendar app notifying the user.

2.4.6.3 Email
Since the phone has a Google account, email to this Gmail address is received by the Gmail app. This is located in the Google folder in the launcher. Enter the menu (top left), Settings. Enter the account (Gmail address). It is recommended to turn off Notifications, unless the email address will be actively used and the primary user wants to be notified of incoming email.

2.4.6.4 Installed apps
The phone comes with many apps pre-installed, some of which can be uninstalled. The following may well be uninstalled to clean up the phone (press and hold icons in the launcher, and drag them to the garbage can at the top right):

- 100% Games!
- Each of the games in the Games folder
- Highlights (In Top apps)
- Zinio (In Top apps)

The app called Backup (in Management folder) should either be uninstalled or have its reminders disabled. To disable reminders, open the app and go to Menu – Settings and set "Backup reminder cycle" to "Don't remind me".

The icons in the launcher can be moved around to reorganize the launcher screens. The widgets on the front page can also be removed. The front page should be set up with the functions most useful to the primary user, including the POSEIDON app.

2.4.6.5 Installing apps
Other apps can be installed on the phone, based on the preferences of the primary user.

Note that SmartTracker can use SMS to send commands to an application, such as to request a position update. Some communication apps which include SMS may consume all incoming messages so that they don't reach our app. We are not planning to use SMS commands for pilot 1, so it is not currently an issue. However, if we start using it, we need to be aware of the issue and remove communication apps which use SMS if there is a problem.

2.4.7 Android/firmware version and updates

There will sometimes be phone system updates available. The Huawei comes with Android 4.4.4, and at the time of writing there haven't been any updates. It is likely the phone will get an update to Android version 5 at some point. This Android version has some major UI changes compared to 4.x, but Huawei also have their own UI modifications on top of Android, which may lessen the visible effect of an Android update.

Phone system details can be found in the settings, "About phone". In the settings, go to "Updater" to check for updates. This should be done before starting the trial, to make sure the phone has the latest version. Then, enter the Updater menu, Update settings, and turn off Auto-check for updates. This is to keep the phone stable during the pilot.

2.4.8 Connecting to development tools

For developers wanting to connect the phone to Android Studio or other tools, an ADB driver is needed on the computer. Our experience is that no special driver was needed on OS X, while on Windows 7 we needed to install Huawei's HiSuite software to get a suitable driver. This can be done by selecting PC Suite mode for the USB connection on the phone, which connects a virtual drive to the computer. This has the installation program for HiSuite. Once installed, the ADB driver should be in place.

As in other Android phones, you need to enable the developer options in the settings. This is done with multiple clicks on the "Build number" field, under "About phone". This allows access to the USB debugging setting.

2.5 Interactive table setup

Having received the interactive table, the proper working of the interactive table has to be checked. For this, the interactive table has to be connected to a screen/monitor through the HDMI input, see Figure 3. Additionally a keyboard can be connected to the USB plug. The interactive table needs to be connected to the internet, so please plug an Ethernet cable from the local router into the table. As the last step, the power plug should be plugged in. This triggers the start of the mini PC from inside the interactive table and also automatically starts the sensor data processing software on the PC.

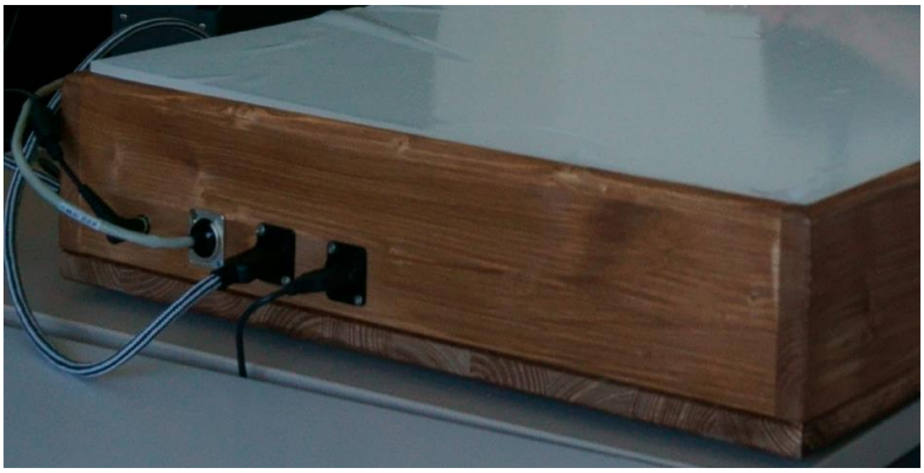


Figure 3 – Plugs of interactive table: power plug, Ethernet jack, HDMI connector, USB plug

Please check that it is properly working. After ~1minute the Ubuntu loading screen should appear on the monitor and you should be automatically logged in as user *captap*.

If this doesn't happen, please open the lid of the table and take a look at the front of the PC, next to the power on button, like shown in Figure 4, there should be a green light, not red, nor blue.

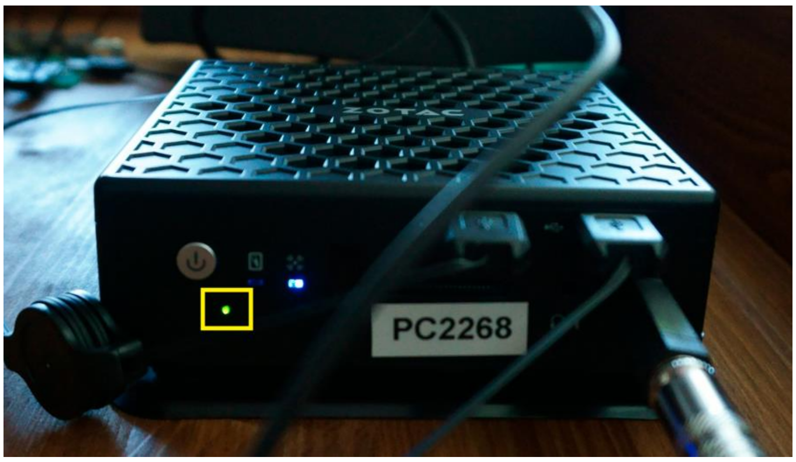


Figure 4 – mini PC from inside the interactive table in power on mode.

A red light means that the power is on but the PC didn't automatically start. This happens if there is a USB hub connected to the USB plug of the interactive table with multiple USB devices plugged to it. Because the PC doesn't have enough power to power all the USB ports, the PC doesn't start. You can solve this by attaching an actively powered USB hub. However, with a single keyboard connected to the interactive table the mini PC started.

An additional check for the functionality of the table by looking under the lid is to check if there is a pulsing light on each of the boards, like shown in Figure 5. If it is pulsing, the boards are successfully communicating with each other, exchanging data. If ever the LED should not pulse, then go to the board marked with a blue sticker and a 0 written on it and plug out the mini USB next to the label and plug it in again. This triggers the communication between the boards to start again.

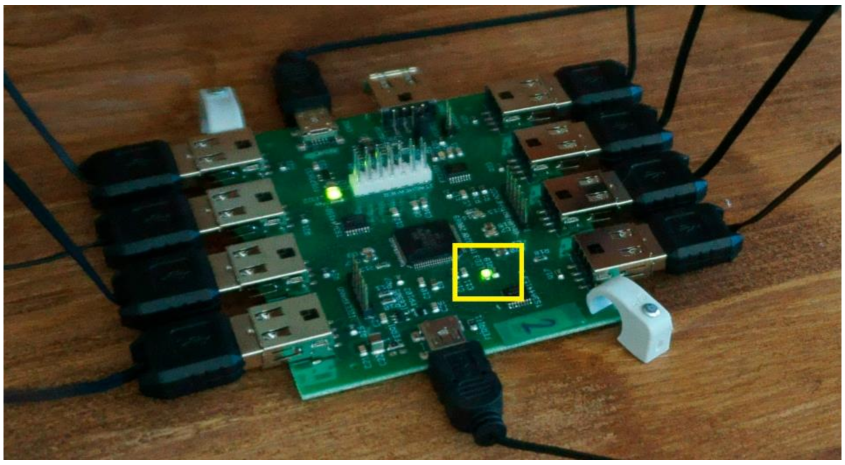
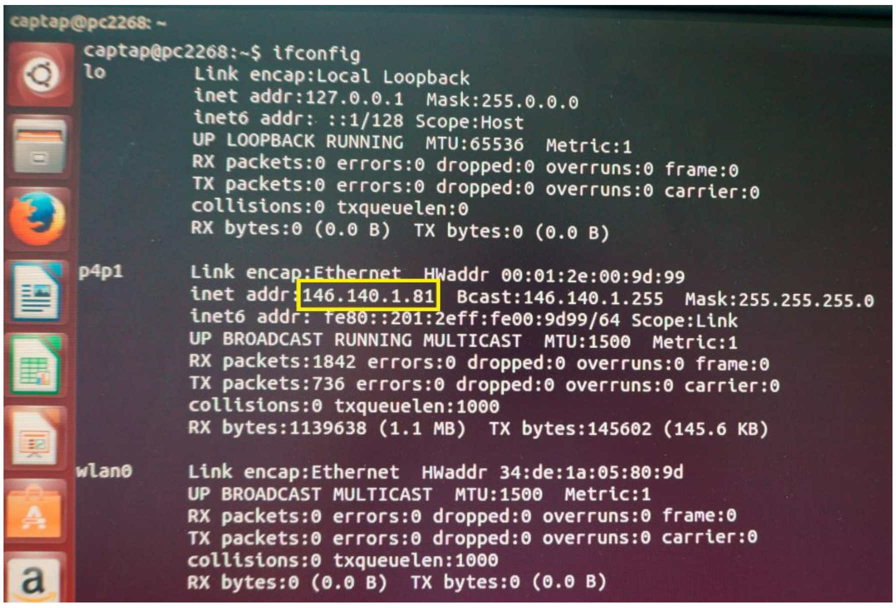


Figure 5 – Check pulsing LED on boards inside the interactive table.

If all of this works and you have the Ubuntu UI, then please press the windows key and then type into the search area *terminal* and press enter, to open a terminal.

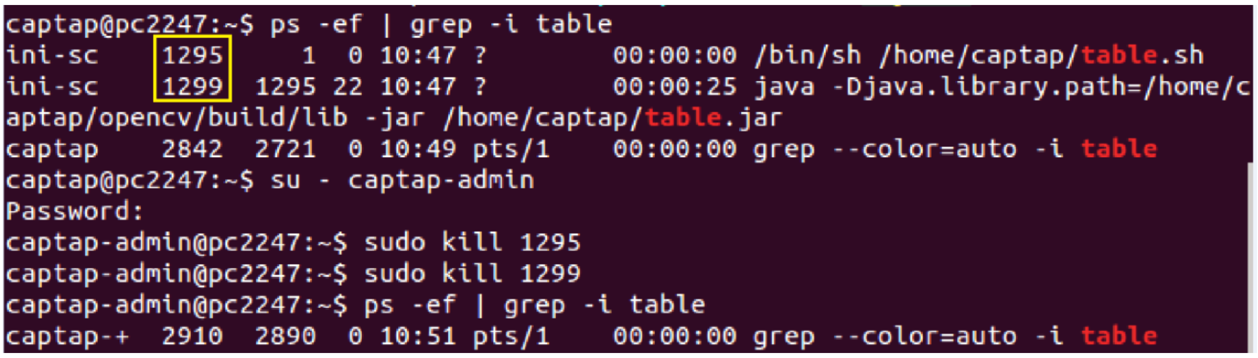


```
captap@pc2268: ~
captap@pc2268:~$ ifconfig
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

p4p1      Link encap:Ethernet  HWaddr 00:01:2e:00:9d:99
          inet addr:146.140.1.81  Bcast:146.140.1.255  Mask:255.255.255.0
          inet6 addr: fe80::201:2eff:fe00:9d99/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1842 errors:0 dropped:0 overruns:0 frame:0
          TX packets:736 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:1139638 (1.1 MB)  TX bytes:145602 (145.6 KB)

wlan0     Link encap:Ethernet  HWaddr 34:de:1a:05:80:9d
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figure 6 – Find IP address of interactive table

First type *ifconfig* to find out the IP address attributed to the interactive table. Later on, you can connect to the table using ssh with this IP address without having to connect an external monitor and keyboard. The IP address is to be found in the second paragraph, like shown in Figure 6. In this case it would be 146.140. 1. 81.

Next, we would like to check if the software processing the sensed data automatically starts. For this we search for the processes which contain the string table by typing *ps –ef | grep –i table*. The output is shown in Figure 7.



```
captap@pc2247:~$ ps -ef | grep -i table
ini-sc     1295      1  0 10:47 ?        00:00:00 /bin/sh /home/captap/table.sh
ini-sc     1299   1295 22 10:47 ?        00:00:25 java -Djava.library.path=/home/c
aptap/opencv/build/lib -jar /home/captap/table.jar
captap     2842   2721  0 10:49 pts/1    00:00:00 grep --color=auto -i table
captap@pc2247:~$ su - captap-admin
Password:
captap-admin@pc2247:~$ sudo kill 1295
captap-admin@pc2247:~$ sudo kill 1299
captap-admin@pc2247:~$ ps -ef | grep -i table
captap-+   2910   2890  0 10:51 pts/1    00:00:00 grep --color=auto -i table
```

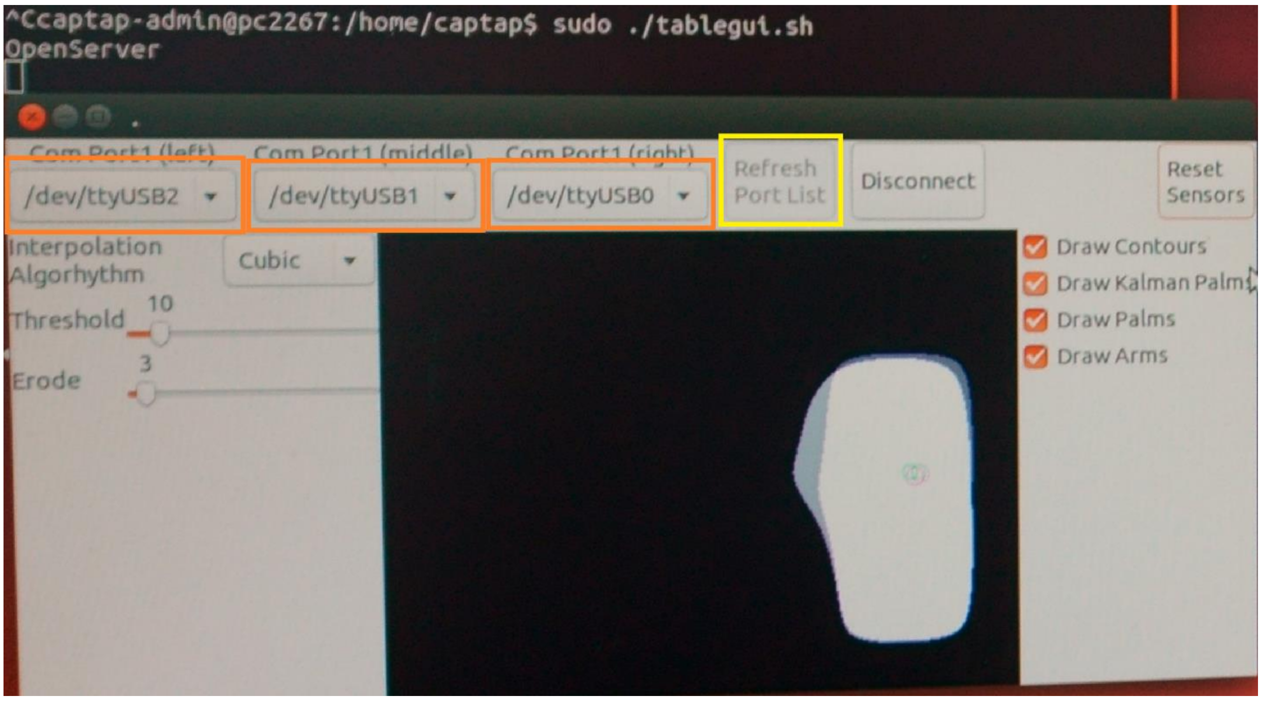
Figure 7 – check if software of the interactive table started

Generally three processes appear to be found. This means that the software has successfully started. In the yellow box in Figure 7 the process IDs are highlighted.

To test the interactive table locally, another program with a graphical interface has to be started. But first, the already running processes have to be terminated. To gain the rights to terminate processes one needs to log in as admin user captap-admin by typing *su – captap-admin*. Then the password is typed in, which is disseminated through the technical partners. By typing sudo kill [processID] the first two processes need to be terminated. The third one does not need to be terminated. By typing *ps –ef | grep –i table*, one can see that the processes does not exist anymore.

To start the software using a graphical user interface go to the folder captap by typing *cd /home/captap*. Now type *sudo ./tablegui.sh* . This command starts the graphical interface shown in

Figure 8. By pressing the button “Refresh Port List”, highlited in yellow, the available ports appear in the drop down menus of the areas highlighted in orange. Please set these areas to ttyUSB2, then ttyUSB1 and ttyUSB0. This ensures that the boards are attributed to the right area of the graphic which appears by pressing the button “Connect”. An irregularly moving white surface appears. This shows the initial state of the sensors. Please pass with the flat hand over the entire surface of the table until there is no white area left after lifting your hand. Now you can place your hand on the top or near the top of the table and a white area will show where the hand has been detected, like shown in Figure 9. Try out also taping on the surface of the table. In the terminal window, the coordinated of the position of the hands will appear, as well the recognized tap will be logged by displaying the string [KNOCK].



```
^Ccaptap-admin@pc2267:/home/captap$ sudo ./tablegui.sh
OpenServer
```

Com Port1 (left): /dev/ttyUSB2
Com Port1 (middle): /dev/ttyUSB1
Com Port1 (right): /dev/ttyUSB0
Interpolation Algorhythm: Cubic
Threshold: 10
Erode: 3

Figure 8 – Graphical user interface of interactive table software

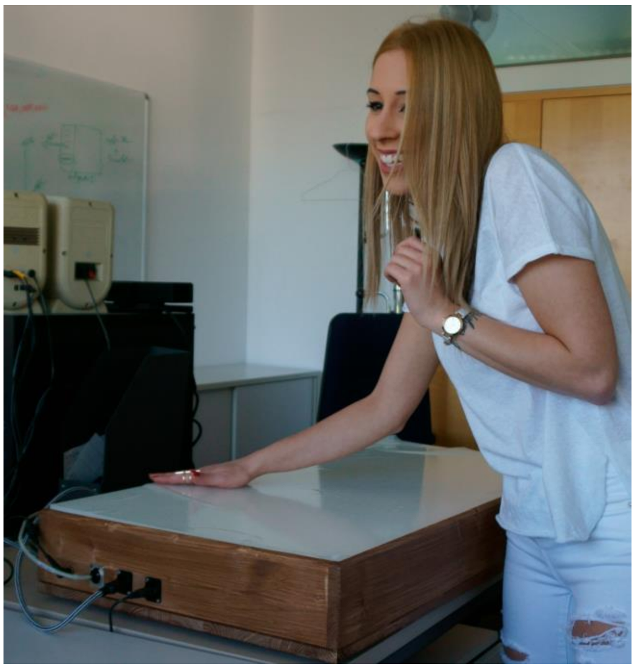


Figure 9 – try out the interactive table

2.6 Computer setup

2.6.1 universAAL Installation

The universAAL middleware needs to be installed on the chosen laptop/PC. To do this, the user needs to download the .zip file containing the universAAL server from the following case sensitive URL: http://goo.gl/XWF9OG . This file should be placed in the main C drive of the user's computer and unzip, leaving a folder named "universaal_server". Next, a shortcut to "C:\univeraal_server\bin\karaf.bat" needs to be created and added to the startup folder, which causes Windows to automatically start the application on startup.

2.6.2 First setup of Moneyhandling App with the interactive table

Before starting the first iteration (Pilot 1) of the Moneyhandling application (Moneyhandling App) you have to do some preparation. First of all connect the interactive table to the local subnet using an Ethernet cable and then plug in the power cord on the side of the table.

NOTE: Please check the following things to be true:

- 1x interactive table stands on a solid table
- 1x computer with network connection and connected to the same subnet as the interactive table
- DHCP enabled (no fixed IP addresses)
- No MAC address blocking

Download the most current version of the Moneyhandling App from http://goo.gl/awqWVX (Windows Version) and extract it to a location of your choice. With the interactive table properly set up and powered on double click the application to start. You will be asked to enter the hostname (of the form PCxyzv), which can be found on the side of the interactive table, like shown in Figure 10. After providing the right hostname the application will connect to the interactive table and you can start navigating and interacting with the app through the interactive table.

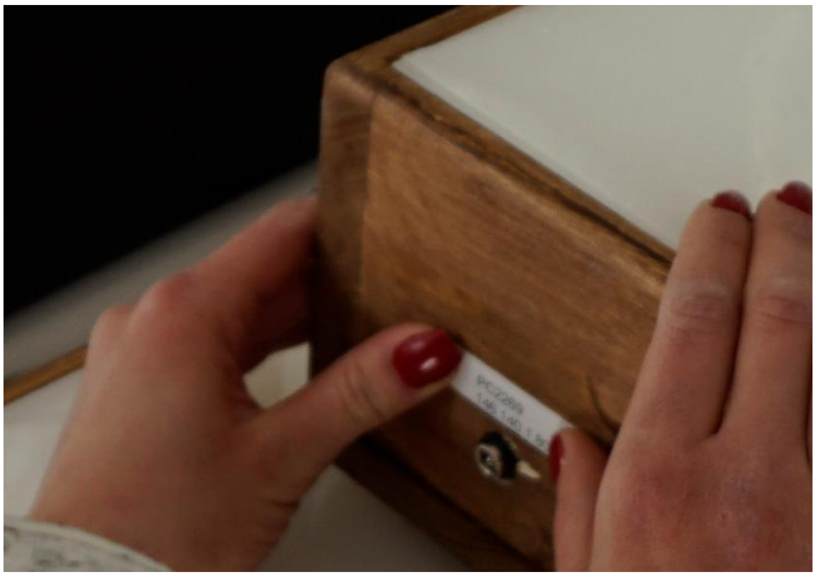


Figure 10 – Hostname of the interactive table

3 Mobile app technical user manual

This is documentation for the prototype 2 iteration of the POSEIDON app. It is meant as a comprehensive, technical documentation, not primarily targeted at end users and including information mainly of interest to technical personnel. An end user version, translated to all project languages, is made by extracting the relevant content.

The prototype mobile application offers navigation, tracking and calendar functionality, and connects to the context awareness middleware, which should also be installed on the device. It will run on any Android device with Android version 4.0 and up, although the user interface is designed primarily for the screen size range 5-7 inches, held in the vertical (portrait) orientation. The primary target device, used in the pilot trial, is the Huawei Ascend G7 phone.

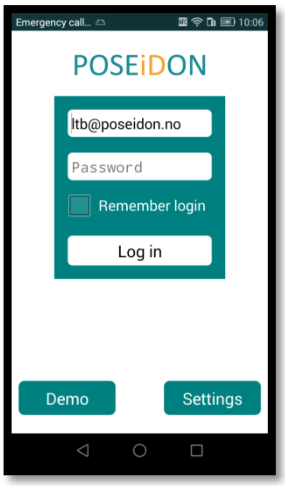
The application is distributed via Google Play. Direct link: https://play.google.com/store/apps/details?id=no.tellu.poseidon

To find it by search in Google Play, use "poseidon tellu" to get a good match.

Make sure to enable automatic updates in Google Play, so that new versions of the application are installed automatically. Also make sure GPS positioning is enabled on the device.

3.1 Starting and modes

A starting screen for logging in is shown the first time the application is started. The connection to an account can be remembered, so this screen is only needed for the initial setup, and doesn't need to be seen again by the primary user. Logging in requires a correctly configured account in the POSEIDON SmartTracker service. User name and password are used for the authentication with this service. Tick "Remember login" to log in automatically the next time the application is started, bypassing this screen. If the login is successful, user data is retrieved from SmartTracker, and the tracking part of the application can send sensor and context observations there, making them available to other POSEIDON applications. The application will also connect to the calendar of the phone, which in turn should be set up to synchronise with Google Calendar.

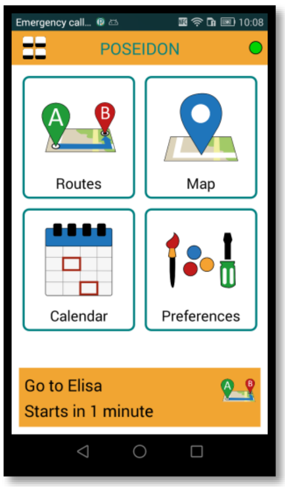

ltb@poseidon.no

The application can also be run in a demo, offline mode. This allows testing and demonstration of the application without the need for setting up an account or connecting to online services. The application will run as normal, but won't send data to SmartTracker or use a real calendar. Instead you get a fixed set of demo content. The application will work without an internet connection, but the map is downloaded from Google, so without an internet connection the map may not be drawn, depending on what map tiles have previously been cached in the device.

Once a real or demo session has been established, the start screen with main menu is shown. The application is not meant to be "turned off" when in real use. When leaving the application with the phone's home button, the POSEIDON icon stays in the notification bar, to show its presence. Except in navigation mode, the only activity it will perform in the background is to check for new calendar events.

Completely turning off the application can be done from the system screen. The Settings are described in the next chapter.

3.2 Start screen and app navigation



The start screen has the main menu for selecting app functions. *Routes* and *Map* are described in the Routes and navigation section of this manual. *Calendar* is described in the Calendar section of this manual, and *Preferences* in the preferences section. When in navigation mode, the two first elements are replaced by *Navigation* (getting back to the navigation screen) and *Stop navigation* (turn off navigation).

Below the main menu, the current or first upcoming event of the calendar is shown, with title and time until start. Pressing it brings up a complete, scrollable list of upcoming events.

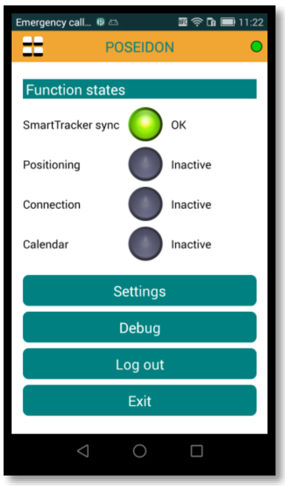
Navigating between the screens of the application, the back button of the Android system, at the bottom of the screen, can be used to backtrack to the previous screen. At the far left on the top bar in the application, there is a start screen icon which will always take you back to the start screen.

3.3 System screen

Currently, the application has a status indicator in the top right corner, which changes colour depending on the state (green – runs as expected, yellow - temporary problem or wait, and red - permanent error). Pressing this while showing the Preferences screen brings up a system screen with functions of a technical nature (the system screen is hard to find on purpose, and you may have to try several times to hit the small status indicator correctly). The end users will not normally need to enter this screen, but it has status useful for diagnostics and local settings such as for GPS simulation.

Buttons on the main system screen:

- Settings: Access to local settings.
- Debug: Detailed status of the tracking system.
- Log out: Exit to the login screen, logging out of the account.
- Exit: Shuts down the application completely (if the login is remembered, this is not lost).



3.3.1 Status information

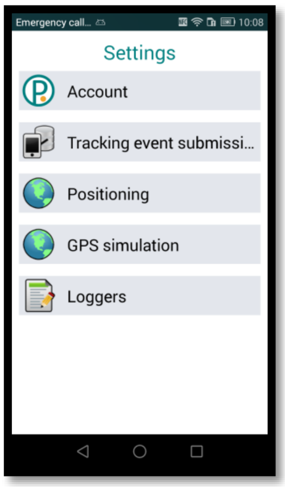
Application status is broken down into contributing components:

- SmartTracker sync: The logged in session and use of the SmartTracker API.
- Positioning: Position tracking, only active when navigation is on.
- Connection: Sending observations to SmartTracker.
- Calendar: Connection to device calendar.

3.3.2 Settings and GPS simulation

The application has a number of technical settings, available from the login and system screens. These are being used in the development process and testing. We will describe the most relevant settings for testing the prototype.

Tracking event submission has settings for sending observations to SmartTracker. If there is a problem with this in online mode, the device id and server URL used can be checked here. The *Positioning* category is for tweaking the position tracking done when travelling.



The most interesting category for testing and demonstration is *GPS simulation*. As it would be very difficult to test route navigation if we actually had to travel the route each time, we have implemented a GPS simulation module, which can be used in place of the real positioning. This can be enabled in the settings, in which case positions will be produced to match the route when navigation is activated. *Simulation speed* is a speed multiplier for the simulation, to go through the route faster than real time. The simulation includes a variable inaccuracy in the values of the position. In addition, based on the *Deviation chance* setting, the simulated movement can deviate from the route. It will then move randomly, and can also switch back to trying to get back to the route. A switch between real and simulated GPS won't take effect while in navigation mode, so switch before starting a route. Note that GPS simulation is independent of the online/demo modes – both real and simulated positioning can be used with both modes, and SmartTracker won't know if a position is real or simulated.

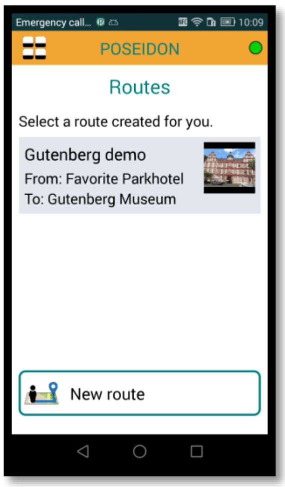
For the developers, the settings also give access to the logging system. All the logs can be sent to a Tellu server for analysis in case of a suspected error.

3.4 Routes and navigation

The POSEIDON application offers navigation using two types of routes. Planned routes are those created with the route planning tool of the stationary system. These are the personalised routes with custom instructions and photos, and should therefore be preferred for navigation. However, it is a fixed route from a specific starting point, and so can only be used when starting from the planned location. To be able to navigate from any starting location and to one of the destinations configured for the user, a new route can be retrieved from a route planning service. This is the approach used by most navigation applications.

3.4.1 Planned routes

Selecting *Routes* from the main menu, planned routes transferred from the stationary system are listed. Selecting one, you can start navigation, being reminded that you need to be at the planned starting point.



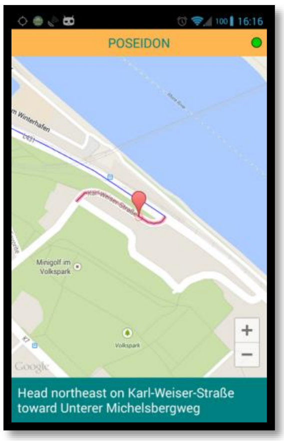
Select a route created for you.

Gutenberg demo
From: Favorite Parkhotel
To: Gutenberg Museum

3.4.2 New route

All destinations configured for the user are listed here, for selecting a destination for navigation. Destinations taken from the planned routes usually include a photo. Additional destinations (without pre-planned routes) may have been specified through the POSEIDON web. Selecting a destination, the application will first attempt to find the current position using the device's satellite positioning system. It will then contact the route planning service. If all goes well and a route is produced, navigation can start.

3.4.3 Navigation and map

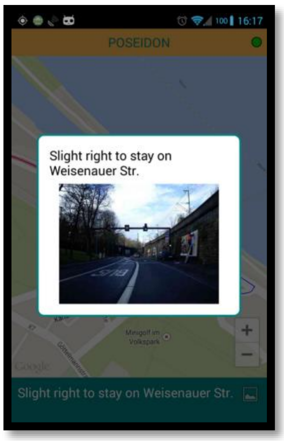

POSEIDON

Head northeast on Karl-Weiser-Straße toward Unterer Michelsbergweg

Starting navigation for a route sets the application in navigation mode. Position tracking is activated, to keep track of route progress and provide route guidance. The route is drawn in the map. The current position (latest position observation) is shown as a red marker on the map, with an accuracy circle indicating the accuracy of the position. The map view uses Google Maps. Map interaction such as zooming and panning is enabled, although whenever there is a new position available the map is centered on this. The map is actually a 3D surface, so it's also possible to tilt the map by swiping with two fingers.

A route is divided into legs, such as for different modes of transport. The application supports both walk legs and transit legs (public transport). A walk leg is further divided into steps, where each step has a textual instruction, and a transit leg can be broken down into intermediary stops. In the map, the current step is highlighted in red.

While in navigation mode, an instruction field appears at the bottom of the screen, always showing the information for the current step. A step can also have a photo description that pops out when entering the step. The photo can be removed if touched and re-added when the instruction field is clicked.


POSEIDON

Slight right to stay on Weisenauer Str.

Slight right to stay on Weisenauer Str.

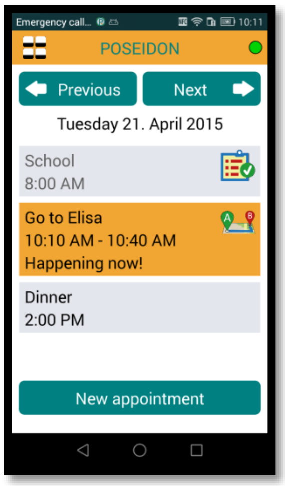
The navigation process stops when reaching the destination. However, the navigation can be stopped at any point by selecting *Stop* option after selecting *Routes* from the main menu.

The navigation algorithm searches forward through the route to find the corresponding place in the route for each position observation. This makes possible finding the correct step even if the start position is located in the middle of the route.

3.5 Calendar and events

The POSEIDON application is also a calendar application. It connects to the calendar data provider in the phone, which in turn has access to the calendars of accounts entered in the phone, such as Google Calendar for a Google account. For the intended usage, linked with other POSEIDON applications, the phone should have a single Google account with a single calendar.

The events in the calendar can be accessed with the *Calendar* function on the main menu. This lists the events of the current day, with title and time. The buttons at the top can be used to change the date, to see the events of other days. Finished events are "greyed out" (text with less contrast), while any current event is highlighted. Press an event to get a detailed view, with any description entered for the event. The most "current" event – started or next to start – is shown on the start screen. This can be pressed for an alternative listing of events, with all upcoming events for several days in a scrollable list.


Emergency call... 10:11

POSEIDON

Previous | Next

Tuesday 21. April 2015

School
8:00 AM

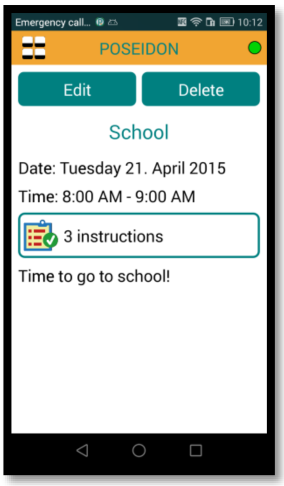
Go to Elisa
10:10 AM - 10:40 AM
Happening now!

Dinner
2:00 PM

New appointment

Events can have two types of additional information, entered on the POSEIDON web. These are indicated with icons on the event item in lists and the start screen, and available from the detail view.

- Instruction list (list icon): An event can have a set of instructions. Each is a short text. These are shown sequentially at the start of an event, and can also be shown from the event detail screen.
- Route for navigation (route icon): An event can include a planned route. At the starting time of the event, the user will be informed. If the user confirms that the app is to provide navigation, it enters navigation mode.

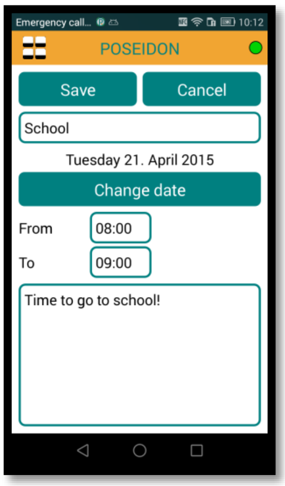


School
Date: Tuesday 21. April 2015
Time: 8:00 AM - 9:00 AM
3 instructions
Time to go to school!

3.5.1 Entering events

The calendar has a button for creating a new event. A title and start time must be entered. An end time can be entered; otherwise the event has no duration. Time can be specified with just the hour, or adding minutes, with or without a colon between hour and minutes. The date is initially that of the date in the calendar when pressing the *New event* button, but may be changed. Events cannot cover multiple dates. A description can be entered to give information about the event.

Note that some of the more advanced features of the web calendar are not available when entering or editing events in the mobile application. It can't be used to enter a recurring event, and a route or instruction list can't be added.



School
Tuesday 21. April 2015
From 08:00
To 09:00
Time to go to school!

3.5.2 Editing and deleting events

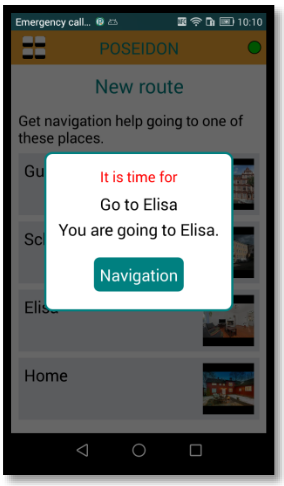
From the detail view of an event, edit and delete actions are available. Edit has the same user interface as for a new event. Title, date, start and end time and description can be edited. Note that it's not possible to edit or delete recurring events. These are events which repeat according to some pattern, and must be managed through a more complex user interface such as that of the POSEIDON web.

3.5.3 Event reminders

An event can have reminders at specific times prior to the start time of the event. This is added to the event in the web interface, and works much like in other calendar applications. The user is given a notification at the reminder time, with the event details and how long it is until the event start time. A notification is placed in the notification bar of the phone if the application is not currently on screen, and the full reminder is shown in the application when it is opened.

3.5.4 Notification at event start

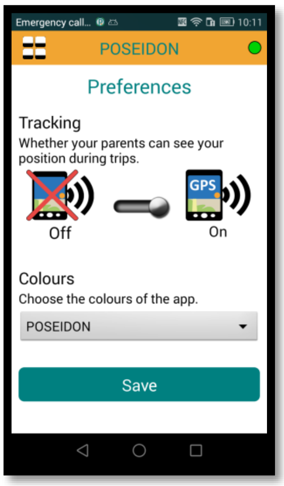
The user is notified of any event at the start time of the event. As with reminders, a notification is placed in the notification bar of the phone if the application is not currently on screen, and a popup in the app gives the details. If the event has instructions, the button on the popup leads to the first instruction, and these are shown in sequence with presses on the button. The rest of the app is unavailable until the notification and instruction dialogs are acknowledged with the button. If the event has a route, the dialog will finally ask for confirmation to start navigation.



Get navigation help going to one of these places.
It is time for
Go to Elisa
You are going to Elisa.

3.6 End user preferences

There are some application and service settings stored in SmartTracker. These can be edited from the POSEIDON web. The colour theme can also be changed in the app, to make this personalization option easily available to the primary user. In addition, the application has its own privacy setting, controlling whether or not position tracking is sent to SmartTracker.


Emergency call... 10:11
POSEIDON
Preferences
Tracking
Whether your parents can see your position during trips.
Off On
Colours
Choose the colours of the app.
POSEIDON
Save

The Preferences screen has the personalisation settings which are available in the app.

- Tracking: When off, position data is not sent to SmartTracker. Any location tracking done will then not be stored or processed on the server, so it won't be available for any researcher or carer to see or for any server-side service logic. Note that the current version will only track position during navigation, so it should not normally be any need to switch this off.
- Colours: This is the theme of the user interface – the colours. The default theme (POSEIDON) is based on the colour palette used in the project. The other theme currently available is a high-contrast theme, with yellow and white on a black background. The colour theme is part of the user account and can be changed from the web.

4 POSEIDON web developer manual

The Poseidon web is part of the POSEIDON project and system. This system is to consist of services and applications to support people with Down's syndrome and, to some extent, also those who interact with them on a daily basis. The Poseidon web is composed open source software component. In this manual we will give a rough view of the techniques used for supporting the HCI interface and system. More details of POSEIDON system development and integration can be found in [1].

4.1 Composition

Poseidon web is supported by the SmartTracker platform provided by TellU AS [2], SmartTracker platform is an intelligent Internet of Things cloud-based service that gathers data from GPS- and other devices. Data is processed by rules and other mechanisms designed to meet the different needs of industries, partners and individual customers. In addition to this platform, we integrated Google Calendar service and third party open source code to Poseidon web. Figure 11 shows an iconized platform architecture of Poseidon web.

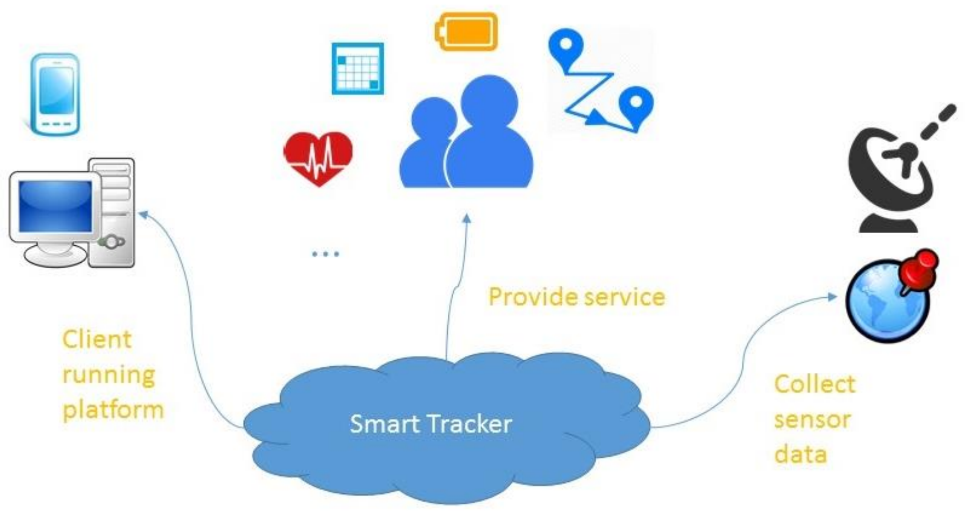


Figure 11 Poseidon web supporting architecture

The Poseidon web is mainly developed by HTML5, angularJS and Jquery, and also integrated some third party open-source code to develop new component. In the following we will introduce two parts of integrated code and what developer can do if they want to study and apply it to the system:

1. Open authorization for public calendar service: In Poseidon web, we choose to use the Google calendar as the calendar publishing provider, this involves the integration of Google service authorization to Poseidon system. We applied the Google Open Authorization 2 (OAuth2) to connect Poseidon web to Google Calendar server. In contrary to tradition authorization methods, the OAuth uses an access token issued by the service provide instead of directly using user name and password to authorize usage of services for third party software. Figure 12 shows the authorization sequence of OAuth using a UML sequence diagram [3].

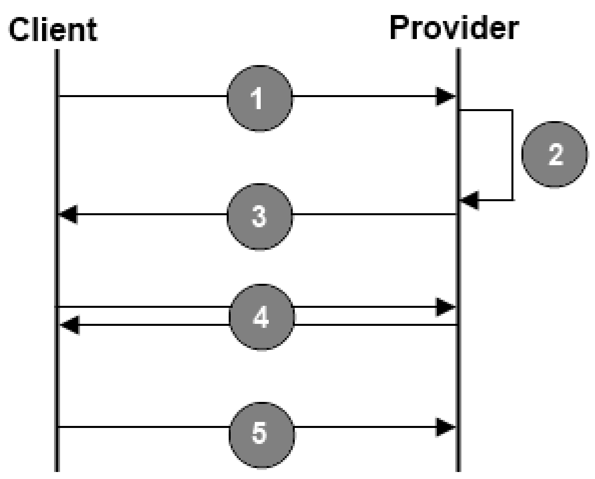


Figure 12 OAuth credential application flow

2. The OAuth credential application as shown in Figure 12 can be described as follows:
   1) The client redirects the user to a login dialog at the provider.
   2) The user authorizes the client.
   3) The provider redirects the user back to the client, additionally returning an access_token.
   4) The client validates the access token.
   5) The access token allows the client to access a protected resource at the provider.

3. Calendar component: The calendar component on Poseidon web is based on Jquery FullCalendar, and encapsulated in an angularJS directive. The FullCalendar API is available on web.

4.2 Open source API

We list the open source APIs in the table below for developers who are interested in extending the Poseidon web system.

API	**Link**
OAuth 2	https://developers.google.com/identity/protocols/OAuth2
FullCalendar	http://fullcalendar.io/docs/
Google Calendar API	https://developers.google.com/google-apps/calendar/v3/reference/events
A FullCalendar AngularJS directive	http://angular-ui.github.io/ui-calendar/

4.3 Conclusions

We will constantly develop and update the Poseidon web during POSEIDON project life-cycle. At present, the system is under heavy development and testing.

4.4 Reference

[1] Development framework, POSEIDON project D5.1.

[2] TellU AS. Link: http://www.tellu.no/

[3] How to Implement Safe Sign-In via OAuth. Link: http://devcenter.kinvey.com/html5/tutorials/how-to-implement-safe-signin-via-oauth

5 Use of calendar data

For pilot 1 there are two applications making use of the calendar. The POSEIDON web is the primary interface for managing the calendar, with the full range of options for creating and editing events. The mobile application is where the calendar events are delivered to the primary user, with reminders, notifications and instructions. For cloud storage and synchronisation of the events between the two applications we are currently using Google Calendar. The calendar data model is as we know from this and other calendar services, with events which may have reminders ahead of time.

This document explains this conceptual model and how the data is used in the system. It is a cross-cutting concern in the system, affecting context reasoning as well as the two applications. This information should also be communicated to the (secondary) end users in some form, to help use the system correctly. For the technical details of the data implementations in Google Calendar and the Android Calendar API, see chapter 3 of D5.4, version 2.

5.1 Calendars

All events belong to a *calendar*. Any Google account has a calendar, and it may have several. An Android device has access to the calendars of Google accounts entered in the device. There may be any number of calendars in an Android device. Our pilot system is meant to be used with a single calendar. The mobile application doesn't distinguish between multiple calendars, but uses whatever is available, so to avoid any confusion it is important that the Google accounts and devices are set up with only a single calendar.

5.2 Event scheduling

The main entity of the calendar model is the *event*. The main attribute of an event is its scheduled date and time. This can get quite complex, as events for most calendar services can be recurring based on some pattern, there can be exceptions to the recurrence, events can have a duration or not and they can be for full days (no start or end time). We do not support all the possibilities of the Google or Android calendar models in our applications. Here is a summary of our rules:

- An event can have a start and end time, or just a (start) time (no duration).
- We do NOT support all day events, or events spanning multiple dates. An event must have a start time, and if it has an end time this must lie on the same date. The reason for these restrictions is to keep the user interface of the mobile app simple, listing events sorted on time for a specific day.
- We support recurring events. This means there can be a potentially open-ended amount of occurrences of an event. Basic patterns of recurrence can be specified in the web interface. Note that any additional attributes of the event, such as description or route, will be the same for each occurrence.

An event may also have a reminder scheduled a specific number of minutes before the start time.

5.3 Other event attributes

These are the other attributes we currently support.

- Title/summary: This is the "name" of the event, and is the primary attribute for identifying the event. It is what is shown in lists, notifications and short descriptions in the mobile app, and should preferably be short enough to fit on a single line on the phone.

- Description: This text can be whatever the user wants to convey about the event. It can be left empty, if the title gives enough information, but will typically give more details about the event. Although there is no explicit limit in the system, it shouldn't be too long – no more than half of the phone screen is a good rule of thumb.
- Route for navigation: In POSEIDON an event can be linked to the navigation functionality by specifying one of the routes planned for the primary user. The mobile app will then offer to start navigation at the start time of the event.
- Instruction list: While the description can be used to give instructions, we also support an ordered list. Each item of the list can be any text, but should be kept short – typically a single, short sentence. Each instruction will be shown in turn in the mobile app, from the notification at the start time of the event (before navigation if the event has both). While this is currently just text, the intention is to be able to add media such as a photo to each item.

5.4 Event notification

The events are communicated to the primary user by the mobile app in two main ways. They are displayed in the calendar so the user can see what is planned for a day, and the ongoing or next event is also shown on the main screen. And the app provides notifications at the scheduled times.

The start time of an event is the main notification time. The mobile app will give a notification with sound at this time. The notification bar of the phone is used if the app isn't on screen at the time, much like other types of alerts such as for received SMS. A dialog box in the app shows the title and description of the event. If there are instructions, the dialog goes on to show these in sequence each time the user pressed the dialog button to advance it. Finally it will ask to start navigation if the event has a route.

If the event has a reminder, a notification is given at the reminder time. It uses the same notification mechanisms as the main notification, but the dialog states the title, description and how much time is left before it starts. Note that a reminder has no information of its own, so except for the minutes left it can't give the user any information different from that of the event itself.

An event is a simple object, and there is no link between events in this calendar model. It is very much up to the secondary users how to use events to create a helpful plan for the primary user. The event and its "start" time is primarily a mechanism for notification. For instance, if the primary user is to be at school at 8:30, we might need an event at 7:50 to remind the user to get ready to go, with instructions on what to bring. Then an event at 8:00 for actually going, possibly with route for navigation. The time to actually be at school – 8:30 – is probably less interesting for this calendar use, but it may be good to enter it so that it is displayed in the calendar in the mobile application.

6 Context middleware

The POSEIDON context middleware is a centralised system for handling context acquisition, management, and reasoning. It can be deployed in two specific scenarios:

- **Centralised Application:** In this scenario, the middleware is a completely separate application which can be installed and used independently. This app is distributed using the Google Play Store[3]. To use the middleware in this scenario, each application needs to use the Service declared as `org.poseidon_project.context.IContextReasoner`. To use this service, each application needs to bind to the service, before interface methods can be invoked.
- **Integrated an Existing Application:** In this scenario, the middleware is integrated as a library into an existing application during compilation. In most circumstances, this usecase is not expected to be used. To use the middleware within this scenario, the containing application needs to use the core service class directly, which is named `org.poseidon_project.context.ContextReasonerCore.` Each request can be made directly, as a conventional java class.

6.1 Using Contexts in Applications

By definition, each POSEIDON compatible application should not have direct control over what ContextObservers are running or not. Instead, the reasoner functions on a need-by-need basis. Only ContextObservers that are actively required by an application are used. If at some point, an observer ceases to be required by an application, it will be automatically stopped.

6.1.1 Declaring a new Context Requirement

When an application requires context information about a particular type, it must declare a new context requirement. This is carried out on the service interface, depending on the usecase scenario introduced earlier. The developer has two specific service methods that can be used for declaring context requirements:

- **addContextRequirement(String appkey, String contextName)**: This method is used in situations where the context does not require any parameters and can be used as it is. This method requires an identifier for the requesting application, and the name of the context passed as method parameters.
- **addContextRequirmentWithParameters(String appkey, String contextName, Map parameters)**: This method is used when a required context also needs specific parameters for it to function. For example, for the Weather context, the system needs to know the specific locations it needs to check the weather for. Similar parameters are required to the standard method described above, except a Map object containing the different parameters needs to be passed also.

6.1.2 Removing a Context Requirement

Once a context requirement has been requested, it will continue to reason and broadcast context updates indefinitely. Therefore, when a given application no longer requires updates for specific contexts, it needs to explicitly inform the middleware. This is carried out by calling the method:

[3] https://play.google.com/store/apps/details?id=org.poseidon_project.context

- **removeContextRequirement(String appkey, String contextName):** This method requires an identifier for the application, and the name of the context it no longer needs passed to it. Once this has been invoked, the reasoner will check if any applications still require the specific ContextObservers involved. If the ContextObservers in question are found to not be required anymore, they are stopped.

6.1.3 Updating Context Parameters

We have described the situation where context parameters might be required by a ContextObserver e.g. Weather. If during the use of a context, specific parameters need to be updated or changed, this can be carried out in the following method:

- **setContextParameters(String appkey, String contextName, Map parameters):** This updates the various parameters of the context in question. This method requires an identifier for the application, name of the context in question, and a Map object containing the parameters passed.

6.1.4 Receiving Context Inferences

Because context management and reasoning is handled externally to each POSEIDON application, we therefore need an approach for applications to receive context updates. To do so, each application must use Android BroadcastReceivers to listen for context broadcasts. These BroadcastReceivers should listen for intents using the name: `org.poseidon_project.context.CONTEXT_UPDATE`. These intent broadcasts will bring the following intent extras that each application will need to collect:

- **context_name**: This is a String that contains the name of the context. An example could be “BATTERY”.
- **context_value**: This is a String containing the value or state of that context. An example for the battery could be “LOW”.

6.1.5 External Context Data Input

Not all context data can be gathered by the context middleware. The centralised context reasoner can include context data sent from other apps in the mobile system. Context values being sent from external apps need to be handled in that application’s ContextReceiver. This means, as the case with context observers, an application needs to have registered its ContextReceiver for context values to be reasoned over. In terms of POSEIDON, we shall have a default ContextReceiver developed as part of our prototypes, which can be extended by other POSEIDON compatible applications.

To send Context values from external applications, this must be carried out using Android Intent Broadcasts. For the ContextReceiver to receive the context, the intent must use the name: `org.poseidon_project.context.EXTERNAL_CONTEXT_UPDATE`. In addition to this intent the following intent extras need to be included:

- **context_name**: A String containing the name of the context source. An example can be “NavState” which is received from the navigation application.
- **context_value_type**: A String containing the name of the internal datatype the value is stored in. An example can be “long”. The ContextReceiver needs this as context data can be many different datatypes.
- **context_value**: A random type which contains the actual context data.

6.1.6 Adding new ContextObservers to the middleware

As the context middleware is intended to be a centralised resource which can be used by multiple applications on the device, the system needs to be extensible with new contexts. These contexts can be added to the middleware by other POSEIDON compatible applications. These ContextObserver classes need to be first in their separate Android .dex file. This can be handled with an Ant or Gradle build script. To import a .dex file containing new ContextObservers, the application in question needs to first copy this .dex file to a temporary location on the SD card, so the middleware can access it. Once this has been carried out the application can call the following method:

- **importDexFile(String appkey, String dexLocation, String[] contexts, String packagename, int permissions):** This method requires the applications identifier, the location of the .dex file, an array of the different ContextObserver names contained in the file, the namespace/package the ContextObservers are contained in, and the usage permissions of the observer. If the permission is set to 0, it is a public ContextObserver that can be used by any POSEIDON compatible app. If however the permission is set to 1, this means the observer is private and can only be used by the application adding it to the middleware.

Following invocation of this method, the .dex file copied to the temporary SD card location can be removed.

6.2 Context from POSEIDON prototype 2 app

The mobile application will send the following context events to the middleware, over an intent broadcast on the intent `org.poseidon_project.context.EXTERNAL_CONTEXT_UPDATE`.

context_name	context_value_type	context_value
NavState	long	0: Off 1: On, critical deviation 2: On, deviation 3: On, no deviation
CalReminder	long	Milliseconds until event start
CalEvent	long	Milliseconds until event end

6.2.1 NavState

An event is sent each time the navigation state changes. Deviation (2) means the user is further from the planned route than some limit and therefore is instructed to return to the route. Critical (3) means they are too far away for the current route to be used for navigation. The user is asked if they want a recalculated route from their current position to the destination, or to quit navigation. In the first case a new route is retrieved and navigation restarted, so the critical state will soon be followed by either 3 or 0.

6.2.2 Calendar events

Context events are sent on the start time of a calendar event, and on the times any reminders are shown before an event. The user is notified at the same time as the context is broadcast (usually posted to Android's the notification bar). The time value for CalReminder could possibly be negative, if the calendar event is found by the app after the start time of the event. The time until the end of the calendar event is 0 if the event has no duration.

7 Route data semantics

This is the technical documentation for the route data, written and used by the home navigation system and used by the mobile app for navigation.

Route data created on the stationary system is a set of files stored in an archive on the local disk. This archive is transferred to the mobile device and the files placed in the local file system, to be used for navigation. This chapter contains the specification of the contents of the archive.

7.1 Structure

The overall structure of a POSEIDON route will be an archive file (e.g. .zip) which will be broken down into the following components:

- POSEIDON Meta Data
- Directions Data
- Media Archive

This file during copying/installation can be copied and unzipped into the following location: /SDCARD/POSEIDON/Routes/$RouteID$/

The $RouteID$ will be stored in the POSEIDON Meta Data file, and will provide us with easy relative locations to use. (Don't need to worry about absolute directories)

7.2 POSEIDON Meta Data

This file will hold the POSEIDON meta data related to a route, which can be stored in the app database.

Filename: meta.json

JSON identifier	Class field	Datatype	Comments
`id`	`id`	`long`	Unique id (from stationary system).
`title`	`title`	`string`	Name for selecting the route in the UIs.
`start_location`	`startPoint.name`	`string`	Name of starting point, to tell the user where they need to start from.
`end_location`	`endPoint.name`	`string`	Name of destination, to tell the user where they will go.
`start_longitude`	`startPoint. longitude`	`double`	
`start_latitude`	`startPoint.latitude`	`double`	
`end_longitude`	`endPoint. longitude`	`double`	
`end_latitude`	`endPoint.latitude`	`double`	
`resource`	`imageURI`	`string`	URI for image representing the route, in media folders.

If a route is not specified with an "id" we will need to just insert into the db, get a new id from the db, and change the folder name to be the same.

7.3 Directions Data

This file will contain the directions in a format similar to Google Directions to keep some compatibility.

Filename: directions.json

Currently, the difference is that in each segment, there can be an array of "resources", with each "resource" pointing to a particular image/video/audio file.

While both Google Directions JSON and the object structure of the mobile app has leg objects containing step objects, it’s not a 1-to-1 correlation between the two data models. The JSON can have a single leg for a route with multiple transport modes, with a step for each mode containing nested steps for detailed instructions. In the app objects, there needs to be one leg for each transport mode, and just one level of steps. So in this case the outer level of steps in the JSON is the app legs and the sub-steps are the app steps.

7.3.1 Leg

JSON identifier	Class field	Datatype	Comments
`(step. travel_mode)`	`mode`	`string`	In each step in JSON data, but the navigation needs it on leg level. See *Travel modes* section below.
?	`legId`	`string`	Short name of leg, such as bus number. Used in transit mode instruction.
?	`legName`	`string`	Long name of leg, such as start and destination of transit line. Not currently used.
?	`headsign`	`string`	Headsign of the bus or train being used. Used in transit mode instruction.
?	`agencyName`	`string`	Name of the transit agency that operates the service. Not currently used.
`duration.value`	`duration`	`long`	Leg's duration. JSON is seconds, class needs milliseconds.
`distance.value`	`distance`	`float`	Distance to travel, in meters.
`start_location` `start_address`	`from`	`object`	Name and lat,lon of leg start. If JSON has full address, extract street address (up till first comma).
`end_location` `end_address`	`to`	`object`	Name and lat,lon of leg end. See previous row. Used in transit mode instruction.

7.3.2 Travel modes in mobile app

App code is so far based on OpenTripPlanner mode names, such as WALK, BUS and TRAM. It is used by navigation logic:

- The main distinction is between active (user controls movement) and transit (user is transported) modes. So far active mode is only true if mode==”WALK”. We can include driving as active mode (we won’t fully support driving instructions, but walking instructions may be used).

- In transit modes, instructions include the mode name, such as “Take <mode> two stops to <stop name>”.

7.3.3 Step

JSON identifier	Class field	Datatype	Comments
`start_location`	`latitude`	`double`	Step class position is start of leg.
	`longitude`	`double`	
`distance.value`	`distance`	`float`	
`html_instructions`	`instruction`	`string`	JSON instruction includes HTML tags, which are removed in the app.
`polyline.points`	`stepGeometry`	`string`	Encoded polyline bean for the path.
`maneuver`	`relativeDir`	`string`	We could read this from JSON, but we need a clear definition of how it is represented in the class and used for navigation.
?	`imageURI`	`string`	?

7.3.4 Navigation instructions in mobile app

In walk (active) mode, each Step should have an instruction text. Google Directions provides basic step instructions, or this may have been changed by route customisation.

In transit mode we have so far only generated instructions based on fixed templates where leg properties are plugged in. We need to support step instructions here as well.

7.4 Media Archive

This will be a folder that contains the different media that will supplement a route. This can be used for storing the media e.g. photos of places for the journey, or annotative text. While for this project we may not use audio/video, it included for structure sake, and it is there if someone ever wants to use it.

Folder name: media

Will contain the following:

- Folder named "p" for Photos/pictures
- Folder named "v" for videos
- Folder named "a" for audio