SmartPlatform POSEIDON developer documentation

This document contains developer documentation for POSEIDON use of the SmartPlatform infrastructure component. It first describes the data flow and data and account model of the platform. It describes the API other POSEIDON components can use to connect to this service. And it describes what data we store in SmartPlatform in the POSEIDON prototype solution. An overview of the SmartPlatform is given in chapter 5 of deliverable D5.1.

Tellu SmartPlatform is a generic and highly configurable platform for data collection and processing. It is used to implement sensor-based services. The core functionalities are:

- Receival of data from a heterogeneous set of sensor devices and protocols.
- The storage of this data into an internal data model.
- Processing of this data by a rule engine.
- A web application for management and data access.
- A REST API to allow access to the data by other services.

This documentation is not a complete developer manual for the Tellu SmartPlatform, but aims to explain what is needed for developers of external applications to be able to connect to the platform. In addition to explaining the account and data model and the main API itself, it will explain aspects of the system that are useful as a background.

SmartPlatform data and interface overview

Figure 1 shows an overview of the platform.

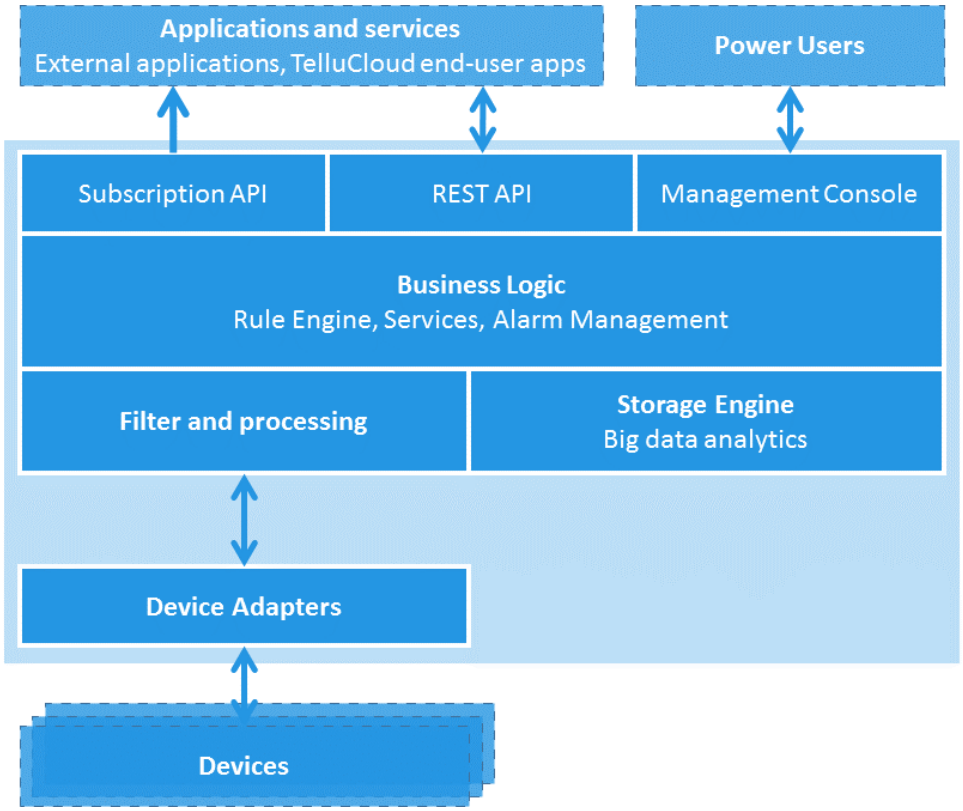

Figure 1: Tellu SmartPlatform architecture overview

There are two different interfaces to the SmartPlatform for other components in the POSEIDON solution – *Device Adapters* (edges) and *REST API* – and it is important to understand how these differ.

Roughly we can say that the first is for input of raw data in forms that are independent of SmartPlatform, while the second gives access to SmartPlatform's own data model, which includes results of the input after processing. So these interfaces are at opposite ends of the system. We will go through these aspects of the system, following the flow of data. But first a quick remark about "devices" and "sensor" data. We use these terms because the SmartPlatform was created for processing data from traditional sensor devices, with inputs such as position, temperature and events registered by the devices. However, as it is a generic platform, the data input can be anything that can be represented with numbers and text. So the "Sensor device" term should be taken in the broadest possible sense.

The SmartPlatform can be set up with Device Adaptors (often called edges) to collect data from sensor devices. The platform can be used with any type of data source by adding an edge that speaks its language and translates incoming data into the platform's internal format. In addition to receiving data from the devices and passing it on to the core, an edge typically supports commands to the devices, such as for configuration, so there is communication both ways. In POSEIDON we have so far only done data collection from client applications we develop within the project, where we have control of the communication protocol. We have a generic SmartPlatform edge where data can be posted using HTTP POST and a JSON format. This simple protocol is easy to implement in our components that needs to post data to the system. However, the edges give the system great flexibility. We can easily connect small purpose-built sensor devices such as a stand-alone GPS.

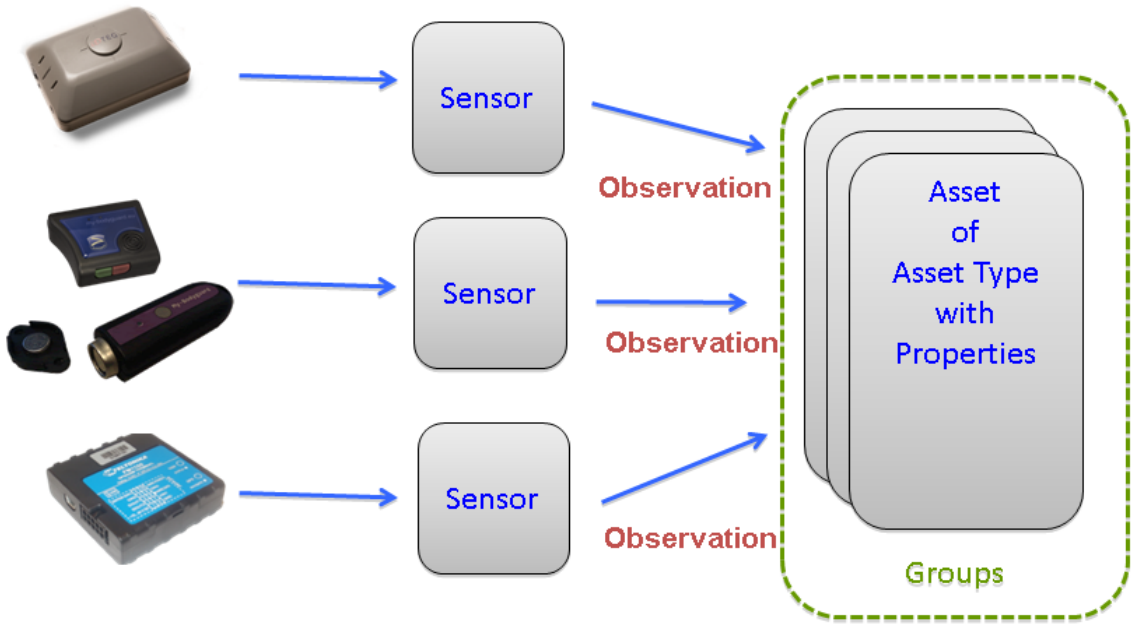

Figure 2: Asset and devices

As this edge part of the system is a sensor device interface meant for data collection, it is independent of SmartPlatform's internal data model. The relevant concepts in the data model are that of *device*, which is a source of data, and *observation*, which is an input to the system from a device. Observation is still called *position* in some places, as initially all observations were positioned, but in addition to some fixed fields such as timestamp and position data they can have arbitrary fields in a key-value map. The central concept in SmartPlatform is *asset*, which is a tracked entity. In POSEIDON, there is an asset representing each primary end user. An asset can have one or several devices, and observations from these devices are then known to be about this asset. An asset may for instance have a position, and it does not necessarily matter where the position comes from, so it

is possible to add a new device to provide position without affecting the service logic that deals with assets and their properties.

Figure 2 shows the relationship between sensor devices and an asset. It is basically the relationship between sources of data and the entity the data is about. The figure also indicates that assets can belong to groups, and that an asset can be of a specific *type*, with the type specifying properties all such assets have. Asset is also the main entity for POSEIDON use of the system. Using fields available for the asset entry, such as properties and tags, we can store context, preferences and other user data, and make it available both for the rule engine in SmartPlatform and for other modules through the API.

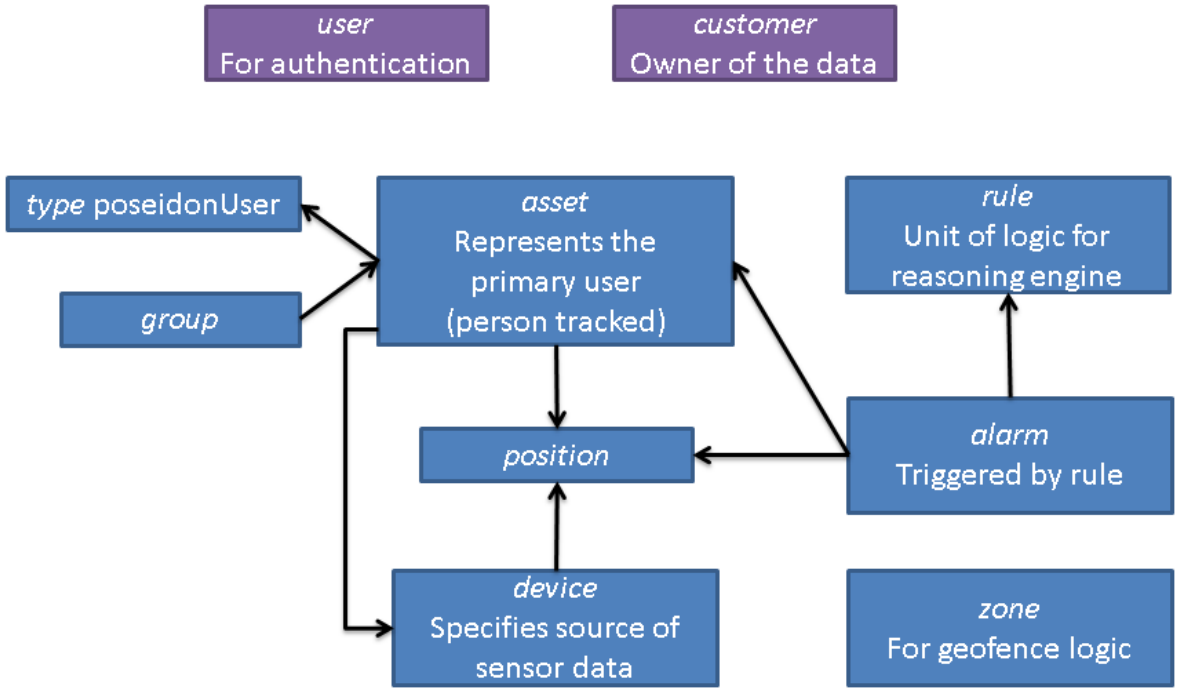

Figure 3: Central SmartPlatform data model entities

Figure 3 shows the most important entities in the SmartPlatform data model, and their relationships (references). Those not already described are mainly of interest to the platform and service itself, for the logic which can be built using its rule engine. An *alarm* entity can be created by the rule engine when some abnormality is detected. If this mechanism is used, alarm entities can be retrieved through the API. The *zone* entity is for creating geofence logic in SmartPlatform, that is, trigging rules based on entering and leaving geographical areas. This can be used to keep track of where a user is (home/school/etc.).

The data model with entities such as assets is stored in SmartPlatform's *Storage Engine* and can be managed through its *Management Console*. Data from edges go through *Filter and processing*, where unneeded data are discarded and the rest is connected to the data model. Every time there is a new observation available from the initial processing, the *Business Logic* (rule engine) processes the change in state, and this can cause rules to trigger, which in turn can update asset state.

The REST API gives access to the objects of SmartPlatform's data model, both for reading and for making changes. The asset acts as a repository of information about the user, and will hold context

such as location, user preferences, and any other types of data that needs to be shared between POSEIDON modules. Fields can have their values updated directly from observations, from the triggering of rules, or through the REST API. Generally, if information may need to trigger rules in SmartPlatform's rule engine, it should be posted as an observation through an edge, otherwise it can be posted directly to the asset through the REST API.

The SmartPlatform also has a *Subscription API*, where external applications can register subscriptions, for instance to all updates pertaining to a specific asset. The platform then maintains an active connection to the subscribing client, rather than requiring the client to poll for data to check for updates. The POSEIDON web application uses this to show the primary user position on a map.

More of the entities of the data model are described in the appendix, in the context of the REST API.

Accounts and authentication

SmartPlatform has an extensive, hierarchical access control scheme, which is described in this chapter.

API access

All access to the system through the REST API requires authentication as a *user* with the right permissions. A user in the SmartPlatform context is someone (or an external system) with access to the system, registered in the system with a user name and password (not to be confused with users in the POSEIDON meaning, although a POSEIDON secondary user may have a SmartPlatform user).

Each API request must include an authentication token, supplied as an HTTP header, which is tied to a user. A token can be acquired in a login transaction giving a user name and password. A token may be time-limited or not. For the other modules in the POSEIDON system, it is possible to issue tokens with no timeout to simplify their interaction with the service.

Account model

Figure 4 shows the main entity types relevant to this data access, where it is important to understand the distinction between account, user and the tracked POSEIDON primary user. At the top of this hierarchy there is a *service provider*, which can be the administrator of a set of accounts. A POSEIDON service provider has been set up for the POSEIDON prototypes. All data available through the REST API is owned by an *account;* this entity type is called *customer* internally and in APIs. Then we have the user, which is just a data access concept as has already been discussed. A user is always tied to a specific account, and can only access the data in this account. But an account can have many users. A user can also have access to the higher levels of the hierarchy. It can be a *service owner*, which means it can manage service providers, or it can be a service provider. A service provider user has access to all the accounts of this service, and can change which account it is accessing through the API.

In addition, there is a very detailed system of permissions which specify exactly which entity types, and which operations on this data, a user has access to. These are collectively managed as *roles*. For instance, an account may have one user with an administrator role, with full permissions to configure the account, and other users with a much more restrictive role to just look at the data.

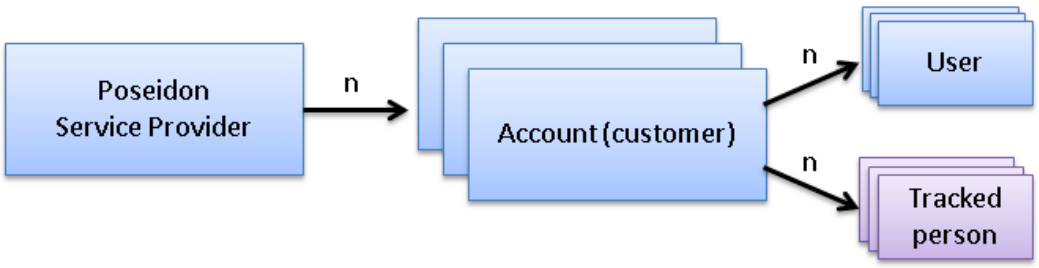

Figure 4: SmartPlatform account model

The POSEIDON primary users are represented as asset entities in the data model, as has already been mentioned. So such a tracked person is “just data” in this context, one of the data objects belonging to an account, and there can be any number of these stored in an account. It is possible to restrict a user’s access to specific assets in an account. All this means it is possible to use a single account for several sets of secondary users each having access only to a specific primary user.

When using the SmartPlatform as part of the infrastructure providing a service to primary, secondary and tertiary users, an account and role policy must be specified. A service provider responsible for this part of the service may set up a separate account for each primary/secondary user pair, or put all users in a single account. If correctly set up with permissions, the end users will not see any difference. The single account model has some important advantages. It makes it easy to give access to the data of a set of primary users, for instance to tertiary users. It also makes it possible to design service logic for the rule engine which considers multiple users, as all rules are account-specific. There is also more work to manage multiple accounts, as entities such as asset type and properties are also all account-specific and must then be duplicated between accounts. For the POSEIDON prototypes, we use a single account for all pilot users.

Device ID

The account and authentication scheme described so far does not concern the Device Adapter part of the system. Each data source (sensor device) has a unique ID in SmartPlatform, and this is what identifies observations as belonging to a specific device entity, which in turn belongs to an asset. A client application instance which will post data through an edge needs to be registered in the SmartPlatform instance, and it needs to know its ID so its posted data will be handled by the system. There are two ways to handle this. One way is to use the device command system supported by edges, where a configuration command is delivered to the device or application instance through the protocol implemented by the edge. The other is to use the REST API, knowing the name of the asset representing the end user, we can retrieve the device IDs for any devices registered for this user.

REST API

The SmartPlatform API is a standard HTTP REST API, supporting GET, POST, PUT and DELETE operations. Data is exchanged in JSON[1] format. This chapter gives a technical description of the format of API requests and replies. The relevant resources (data objects) are described in the appendix.

Note that the documentation for the API is available online at the following URL:

[1] http://www.json.org/

https://telludoc.atlassian.net/wiki/display/SMARTTRACKER/Smarttracker+API+v3

Refer to this for the latest version, and for examples of the JSON data objects.

URL

The URL consists of four parts: base (server address), customer, resource and ID.

```
<base url>/<customer id>/<resource>/<resource id>
```

Retrieving only the root of the URL (without resource) will give an object describing what resources are available.https://telludoc.atlassian.net/wiki/display/SMARTTRACKER/Smarttracker+API+v3

Property	Description
providers	A list of service providers the user making the request can access.
customers	A list of customers the user making the request can access.
access	A list of available resources with a map per resource indicating what methods that is available and whether the client is allowed to perform them.
features	A list of suggestions to the client to enable or disable features in order to provide a simpler interface to the user.
user	An object containing the id of the user making the request.
provider	An object containing the id of the service provider of the customer in the request.
customer	An object containing the id of the customer in the request (or if the customer id was not included in the URL, the customer associated with the user).
time	The time the request was handled.

Retrieving data

All data requests must be done with the HTTP method GET. All requests done on resources will have the same properties in the response.

Property	Description
result	A list of resources matching the data request. This will always be a list, even if the client requests a resource with a specific id.
total	The total number of resources matching the data request.
offset	Marker to use to paginate the data.
max	The maximum number of resources in each response.
user	An object containing the id of the user making the request.

provider	An object containing the id of the service provider of the customer in the request.
customer	The customer used as source of data in the request.
time	The time the request was handled.

Filtering data requests

The SmartPlatform has a powerful filtering mechanism. Filters are added as parameters in the URL. Multiple filters can be added, but a mechanism can only be used once per property (latitude:less=59 and latitude:greater=58 is possible, but name:contains=e and name:contains=m is not). All filters follow the same pattern.

```
<property name>:<filtering mechanism>=<filter value>
```

Filtering mechanisms	Description
equals	Usable on most data types.
contains	Usable on string data types and some more complex types.
less	Usable on number and date data types.
greater	Usable on number and date data types.

Example	Limit request to resources with ...
name:equals=Demo	name equal to Demo.
name:contains=em	Name containing the text “em”.
latitude:less=59	latitude less than 59.
longitude:greater=11	longitude greater than 11.
timestamp:greater=2013-04-18T00:00:00.0	timestamp after April 18. 2013.
timestamp:less=2013-04-20T00:00:00.0	timestamp before April 20. 2013.

Data content

When requesting data, not all data is included due to performance and bandwidth reasons. When querying a list of data, only id and name is included by default. When querying a single resource, all immediate properties are included (without any recursion). Complex objects will (usually) include an id and name. This behaviour can be overridden by adding a parameter to the URL named select. Select accepts a list of property names separated by the character “+”. It also has two special values, star “*” and at “@”. The symbol “*” includes all properties and all subproperties. The symbol “@” includes all properties but only the minimum of subproperties (id and name).

Example	

select=*	All properties of the resource, and all subproperties
select=@	All properties of the resource, but minimum of data for subproperties
select=lastValidPosition+type	Only lastValidPosition and type properties. Type (a complex type) will only have id and name.
select=lastValidPosition+type.icon	Only lastValidPosition and type properties. Type will now also have icon as well as id and name.
select=type.*	Only type. All properties of type will be included.
select=positionProvider.@	Only positionProvider. The immediate properties of positionProvider is included.

Submitting data

Adding a resource must be done with HTTP method POST, without a resource ID in the object or in the URL. Resource objects are wrapped in a list to allow creating more than one object in the same request.

POST `<base>/<customer id>/<resource>`

Updating an object must be done with HTTP method PUT, with a resource ID in the URL. In both cases the resource must a JSON object inside a JSON list in the request payload. See each resource section for more information about which properties that are required and valid values. The resource object is wrapped in a list to be consistent with creating an object. If a property is omitted then it will not be changed on the server.

PUT `<base>/<customer id>/<resource>/<resource id>`

Deleting data

Deleting data must be done with HTTP method DELETE with a resource ID in the URL. The response if successful is an empty GET response (with HTTP code 200).

DELETE `<base>/<customer id>/<resource>/<resource id>`

Tracking and logging history

This section describes the control of storage of tracked data history in the SmartPlatform service, and how the data can be accessed. This is done through the management console, by a developer, researcher or other personnel with an administrator account in the system.

The *Devices* category in the SmartPlatform account shows the last observation that has been made. *Accumulated Observation Values* returns the latest value for various properties. In the *Device* view, the *Debugging Tools* button shows a list of the latest observations, the list can be expanded when blue 'i' icon is clicked.

When the *Personnel* item in the *Content* menu is clicked, the asset properties (with current values) as well as the last observation from the associated device is shown. There is also a button for the History function. When tracking of the person is enabled, all asset property changes are stored, and will be available in this history view, as a table, graph or on a map, depending on the type of data.

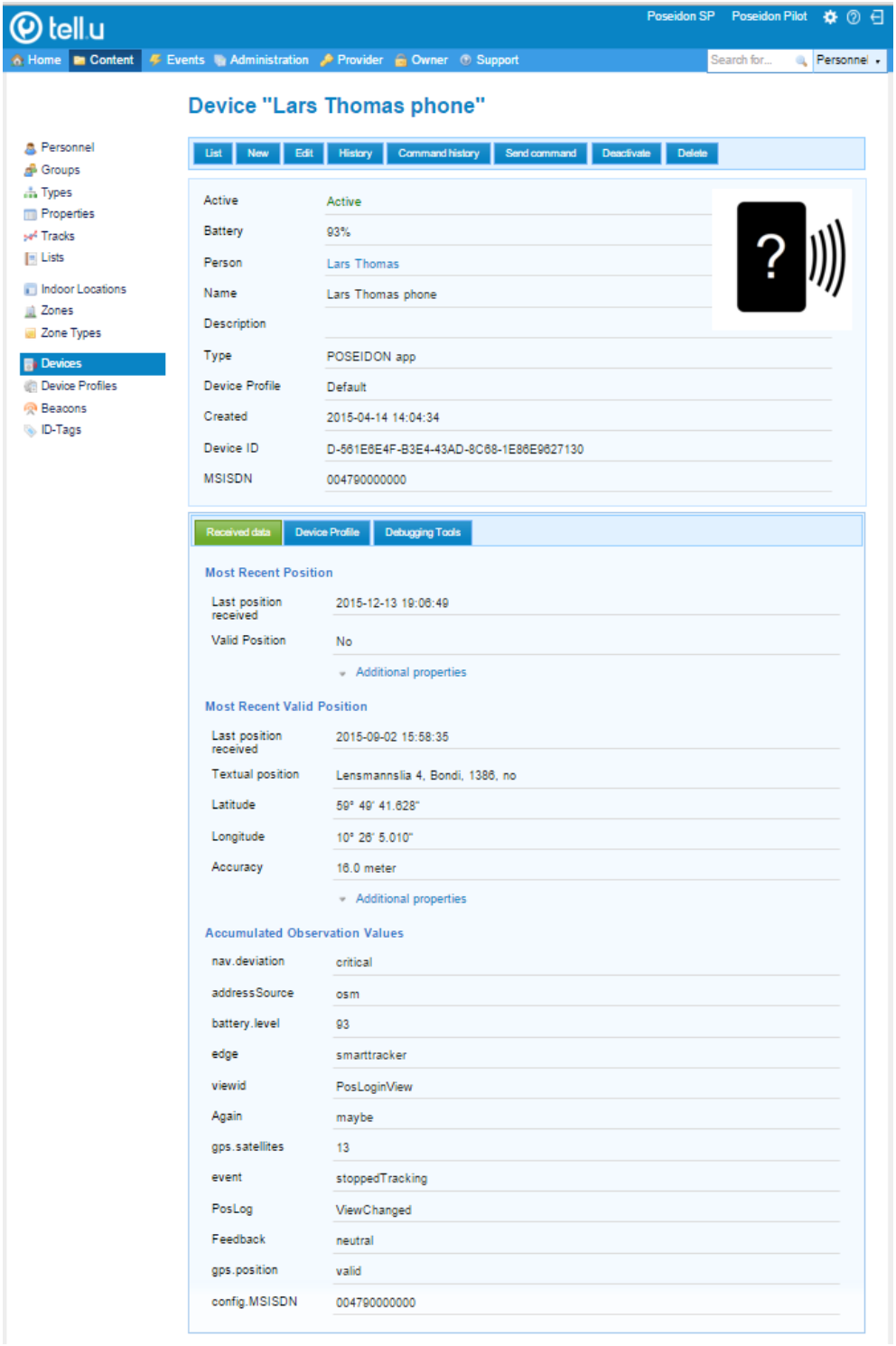

Figure 5: Device view in management console

Data is loaded for a specific timeframe. Tracking, in the sense of storing historical data, can be on or not, and how long the data is stored is configurable. It is not needed to simply keep track of where a person is at the current time. It is needed if a history of movement may be wanted. It also allows using the SmartPlatform service to log events from client applications. In the POSEIDON pilots the mobile application logged end user events to the SmartPlatform server, so that researchers could analyse the logs after the pilots, seeing how much various functions were used. Therefore tracking of history was enabled.

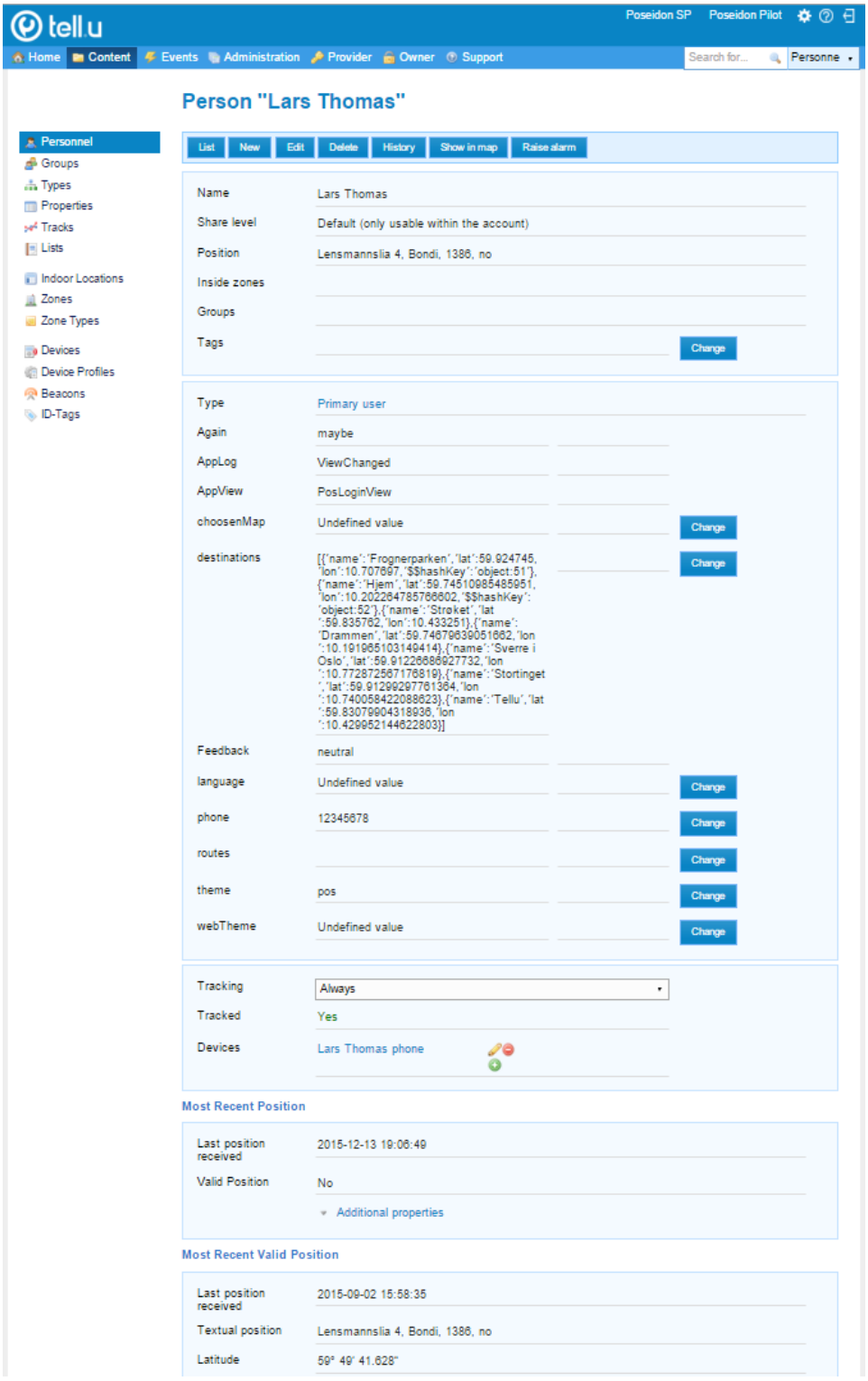

Figure 6: Asset view in management console

Tracking history storage can be affected by one of the following settings:

- The service provider level specifies default and maximum history lengths for accounts belonging to this provider. In the POSEIDON prototype this length is 90 days.
- Any account can set a history length within the service provider limit mentioned before. Thus, we sat the history length to 90 days in the pilot account (POSEIDON Pilot).

- History is only stored when *Tracked* property of the individual asset (tracked person) is enabled. However, tracking can be toggled through the management console, through a rule or through the REST API. *Tracking* option can also be set to *Never* or *Always*.

All of the above are related to storage of data on the server, and should not be confused with the tracking setting in the user preferences in the POSEIDON prototype mobile application. When this is turned off, device tracking isn't sent to the server in the first place.

POSEIDON asset type and properties

The asset representing the POSEIDON primary user in the SmartPlatform service needs to be of an *asset type* defining a set of properties. These properties makes up a user profile. This is a place for applications to store shared preferences, such as for personalisation. An asset property is also needed for observation data to be stored in asset history.

The POSEIDON asset type is called "Primary user". Figure 7 shows the properties defined at the time of the second pilot, in the management console. Some of these are used by specific applications to store their own preferences, and so not part of the framework.

Properties

Add Remove

Property	Name	Property type		Value type	Default Value	Unit	
Feedback	Again	From device	Again	Text	-	-	
AppLog	AppLog	From device	PosLog	Text	-	-	
AppView	AppView	From device	viewid	Text	-	-	
Feedback	Feedback	From device	Feedback	Text	-	-	
choosenMap	choosenMap	User input		Text	osm	-	
destinations	destinations	User input		Text	-	-	
language	language	User input		Text	en	-	
phone	phone	User input		Text	-	-	
routes	routes	User input		Text	-	-	
theme	theme	User input		Text	pos	-	
webTheme	webTheme	User input		Text	default	-	

Figure 7: Asset properties in POSEIDON pilots

Those properties which have an interest outside of a specific application are described in the table below. The framework allows extending the type with more properties as needed.

Property name	Description
destinations	A set of destinations the primary user may want navigation assistance traveling to. Used by the mobile application to create a new route to a wanted destination. It is stored as a JSON list, where each element has the properties name, lat and lon (the two last are the coordinates).
language	Language preference, currently used by the web application.
phone	The phone number of the carer to contact if the primary user needs assistance.

theme	The visual theme for the primary user. Currently supported are “pos” and “posHC”, for the default POSEIDON theme and a high-contrast alternative.
webTheme	Visual theme for the secondary user (used by the web).
Feedback	For logging from the mobile application – feedback on the use of app functionality.
Again	For logging from the mobile application – whether the user would like to use the functionality again.
AppLog	For logging from the mobile application – application event.
AppView	For logging from the mobile application – a view is shown in the app.

Appendix 1: SmartPlatform API resources

We include here a documentation of the most important resources (data objects) of the data model and REST API of Tellu SmartPlatform. The resource sections are ordered alphabetically, and list all fields available through the API. The complete REST API documentation is available online, and should be consulted for application development, as it gives a complete and up-to-date listing. It is available at the following URL:

https://telludoc.atlassian.net/wiki/display/SMARTTRACKER/Smarttracker+API+v3

Alarm

An alarm is a notification that requires the attention of a user, usually generated by the reasoning engine based on some rule.

Property	Type	Optional	More info	Filtering
name	string	-		-
owner	customer	-		-
dateCreated	date	-		equals, greater, less
lastUpdated	date	-		
comment	string	-		equals, contains
ackNeeded	boolean	-		equals
logLevel	integer	-	Degree of severity of alarm. 0 is most severe, -20 least. -20 should be without immediate notification.	equals, greater, less
asset	asset	-	Asset associated with the alarm.	equals (id of asset)
ackedBy	user	-		
rule	rule	-		
alarmCenter	customer	-		
trigger	position	-	The observation triggering the rule/alarm, if available.	
position	position	-	The position of the related asset when the alarm was created.	
zone	zone	-	Zone relevant to the triggering of this alarm.	-

Asset

An asset represents a person tracked by the system (the primary user). The *properties* and *tags* fields are the most important to the other modules in POSEIDON, as these are where the user data and state is stored.

Property	Type	Optional	More info	Filtering
name	string	NO		equals, contains
description	string	YES		equals, contains
owner	customer	-		-
lastValidPosition	position	-	The most recent observation with a valid position received by the position provider of the asset.	-
lastPosition	position	-	The most recent observation received by the position provider of the asset. If the position is valid this will be the same as lastValidPosition.	-
tags	list of objects	YES	When creating or updating only the name of the tag will be used, the time will be set by the server.	contains
groups	list ofgroup	YES	When creating or updating only the id of the group will be used.	-
insideZones	list ofzone	-		contains (id of zone)
icon	string	-	This icon is the icon set by the asset's type.	-
image	string	-		-
type	type	YES	When creating or updating only the id of the type will be used.	equals (id of type)
tracked	boolean	YES	If enabled all observations received by the position provider will be stored. Cannot be set to true if the trackMode is "never", cannot be set to false if the trackMode is "always".	-
trackMode	string	YES	"always" will always store observations received. Rules	-

Property	Type	Optional	More info	Filtering
			cannot change whether or not the asset is tracked. "never" will never store observations received permanently. Rules cannot change whether or not the asset is tracked. "manual" depend on the tracked property to determine if observations are stored. Rules can change whether or not the asset is tracked.	
properties	list of objects	YES	The possible property names are based on the configured properties in the asset's type.	contains
alarms	list ofalarm	-	The five most recent, unacknowledged alarms. Useful for creating lightweight clients.	-
deviceCommands	list of objects	-	TODO	-
positionProvider	tag ordevice	YES	When creating or updating only the id of the object will be used. The server will first attempt to find a device matching the id and if not found a tag.	equals (id of positionProvider)

Device

A device specifies a source of sensor data, and is assigned to an asset to provide sensor data for that asset. In POSEIDON the app running on the user's tablet will be one such "device", and any other client sub-system posting data through an edge will also have a device entry.

Property	Type	Optional	More info	Filtering
name	string	NO		
description	string	YES		
owner	customer	-		
lastValidPosition	position	-		
lastPosition	position	-		
sensorDeviceType	string	NO		

Property	Type	Optional	More info	Filtering
active	boolean	-		
uuid	string	-		
primaryProperties	object	NO		
commandProperties	object	-		
additionalProperties	object	YES		
filters	list of filter	YES		

Group

Assets can belong to groups, which may be useful for group logic (trigger rules for all assets in a group).

Property	Type	Optional	More info	Filtering
name	string	NO	Any non-empty string. Cannot be the same as any existing group inside the customer.	equals, contains
description	string	YES		equals, contains
owner	customer	-		
assets	list of assets	YES		

Position

Although called position, this is more generally a sensor observation. The *properties* field can contain whatever key-value pairs the source wants to post.

Property	Type	Optional	More info	Filtering
asset	asset	-		equals
valid	boolean	-		equals
latitude	double	-		equals, greater, less
longitude	double	-		equals, greater, less

Property	Type	Optional	More info	Filtering
accuracy	integer	-	Estimated accuracy in meters	equals, greater, less
speed	integer	-	Speed as reported by device in meter per second.	equals, greater, less
address	string			-
timestamp	date			equals, greater, less
properties	object			-
events	list of string			-
insideZones	list ofzone	-	A list of zones the position was inside when it was received by the system.	contains (id of zone)

Rule

A rule is a configurable unit of logic for the reasoning engine. The active set of rules defines the service behaviour. The data available through the API is mainly for viewing and changing rule states (turn on and off).

Property	Type	Optional	More info	Filtering
name	string	-		equals, contains
description	string	-		equals, contains
owner	customer	-		-
status	string	YES	Values: "active", "inactive", "unknown", "stopped", "failed", "requiresConfiguration". When changing the status, only "active" or "inactive" are valid inputs.	-

Type

Assets can be typed, with the type specifying what properties an asset has. POSEIDON assets will have their own type, specifying the fields needed for our system.

Property	Type	Optional	More info	Filtering
name	string	NO	Any non-empty string. Cannot be the same as any existing type inside the customer.	equals, contains
description	string	YES	Any string.	
owner	customer	-		
icon	string	YES	URL referring to an icon describing assets of this type.	
properties	object	YES	Object where each key is a different property name. Property value is an object with at least two entries, type and valueType. If the type is a list it must also include a list of string called items.	

Zone

Zones are used to define location-specific logic such as geofence (trigger a rule on entering or leaving an area).

Property	Type	Optional	More info	Filtering
name	string	NO	Any non-empty string. Cannot be the same as any existing zone inside the customer.	equals, contains
description	string	YES		equals, contains
owner	customer	-		
position	latlon	-	An object with two entries, latitude and longitude.	
singleLevel	boolean	YES	When checking if an asset is inside, do they need to be on the same floor.	
floor	integer	YES		
textual	string	YES		
address	string	YES		
points	list of latlon	NO	List of objects with two entries, latitude and longitude. Must have at least 3 objects.	

Property	Type	Optional	More info	Filtering
assets	list ofassets	-	A list of assets that have received an observation with a valid position after this zone was created.	