Smart Anything Everywhere Initiative

Area 3: Advanced micro-electronics components and Smart System Integration Project: H2020–No 761809

diatomic

Digital Innovation Hubs boosting European Microelectronics Industry

Health Experiment
Step-by-Step Integration

Author(s):	João Quintas, Sérgio Sousa, Matthias Schneider, Rubén Hermoso
Status -Version:	V1.1
Date:	02 March 2018
Last update:	23 March 2018
Distribution - Confidentiality:	Public
Code:	DIATOMIC_Health_Experiment_Step_by_Step_Integration_FINAL.docx

Abstract:

This manual provides a useful step-by-step guide for the extended eVida VFK My Signals integrated platform.

Disclaimer

This document may contain material that is copyright of certain DIATOMIC beneficiaries, and may not be reproduced or copied without permission. All DIATOMIC consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

The DIATOMIC Consortium is the following:

Participant number	Participant organisation name	Short name	Country
01	INTRASOFT International S.A.	INTRA	BE
02	F6S NETWORK LIMITED	F6S	UK
03	BioSense	BIOS	SRB
04	Synelixis Solutions	SYN	EL
05	Instituto Pedro Nunes	IPN	PT
06	Fraunhofer IPA	IPA	DE
07	InoSens	INO	SRB
08	Libelium Comunicaciones Distribuidas SL	LIB	ES
09	FastTrack	FASTT	PT

The information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

Document Revision History

Date	Issue	Author/Editor/Contributor	Summary of main changes
02.03.18	V0.1	Sérgio Sousa (IPN)	Initial version of the document
09.03.18	V0.2	Matthias Schneider (IPA)	VFK related content
13.03.18	V0.3	Rubén Hermoso (LIB)	MySignals related content
14.03.18	V0.4	Sérgio Sousa (IPN)	Document formatting and review
15.03.18	V0.5	Sérgio Sousa (IPN), Rubén Hermoso (LIB)	Document structure changes
15.03.18	V1.0	João Quintas (IPN)	Review and acceptance
23.03.18	V1.1	Sérgio Sousa (IPN)	New section 2.1.3

Table of Contents

List of Abbreviations

API	Application programming interface
GUI	Graphical user interface
HL7	Health Level Seven International
IPA	Institute for Manufacturing Engineering and Automation
IT	Information technologies
JSON	Graphical user interface
MSB	Manufacturing Service Bus (middleware supplied by VFK)
PHR	Personal Health Record
REST	Representational State Transfer
RIM	HL7 Reference Information Model
SCM	Services Cloud Manager
VFK	Virtual Fort Knox

1 Introduction

This document presents a thoughtful guide on how to integrate an existing solution with DIATOMIC platform. This platform contains several components – described in detail later – which can be integrated separately. Users are not complied to integrate their solutions with all the components described here. They can choose to integrate with the ones that best suit their needs.

For the purpose of this document, a full integration with all the components is described, so users can have a broader view of all the potentialities of every component and the greater value added by integrating then together.

1.1 Goal

Offer end users a cost effective platform for medical devices that could be utilized as an alternative for measuring different health parameters without the need of a big investment in equipment.

1.2 Test scenario

Here a test scenario is presented. The goal is to have the sensor medical data acquired, stored in HL7 format, and easily available for viewing. Figure 1 shows the conceptual flow for the integration of the different components.

Figure 1: Components integration

Medical data is acquired by the means of a sensor integrated with MySignals. MySignals should retrieve user information from eVida **(1)**. After that it send the sensor and user related data to VFK **(2)**. VFK must them work this data into HL7 format and store it, so eVida can get it **(3)** for easy display through the PHR interface.

2 Modules description

In this section, the different modules are presented and described. It is important to get familiar with all the modules, as some preliminary steps on how to configure / setup the different them maybe required. These instructions are important and should be followed in order to have a smooth integration.

2.1 eVida

eVida's main objective is to provide the infrastructure and tools to quickly develop new applications and deliver them to end-users. Therefore, it provides an answer to patient health self-management supported by technology (https://www.youtube.com/watch?v=fwHnHbcpVHY). The platform already provides an electronic personal health record and supports remote monitoring from devices.

2.1.1 Create a new account on the eVida platform

The first step is to create a user account on the eVida platform. Go to https://www.evida.pt and register an account. As a logged user you can access the platform's web store.

By default, when a user access for the first time to the platform, the account is not set to a developer account. To be able to develop applications for eVida it is needed to activate the developer option (Figure 2). Log in, go to *Settings (1)* in the sidebar, then *Developer Settings (2)* and toggle the option **Connect this account to a developer account** to **YES (3)**.
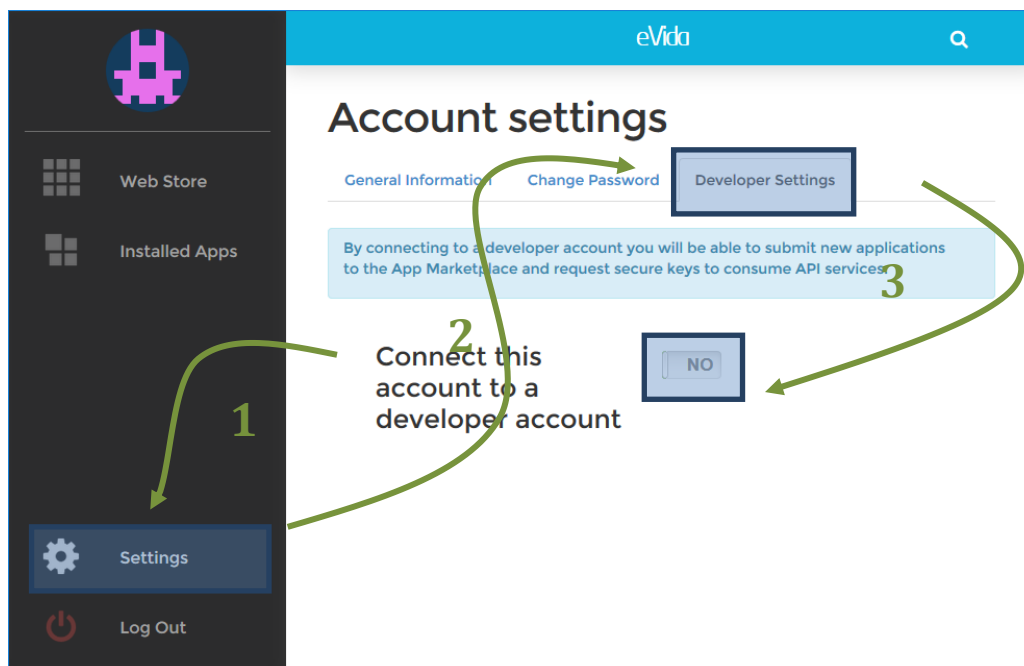
Figure 2: Connecting account to a developer account

2.1.2 Create an OAuth Consumer

Authentication in eVida platform is done using *OAuth2*. This is an important step as it will allow the retrieval of information, such as the user logged in.

To perform the *OAuth2* flow, a **Consumer Key** is needed, which identifies your application. To get the **Consumer Key**, go to *Developer (1)* section and then access *API access (2)* tab. Once there, click *Create another Oauth Consumer (3)* (Figure 3), fill the form – the field Redirection URI may be left blank – and submit it. A **Consumer Key** and **Consumer Secret** will then be generated.
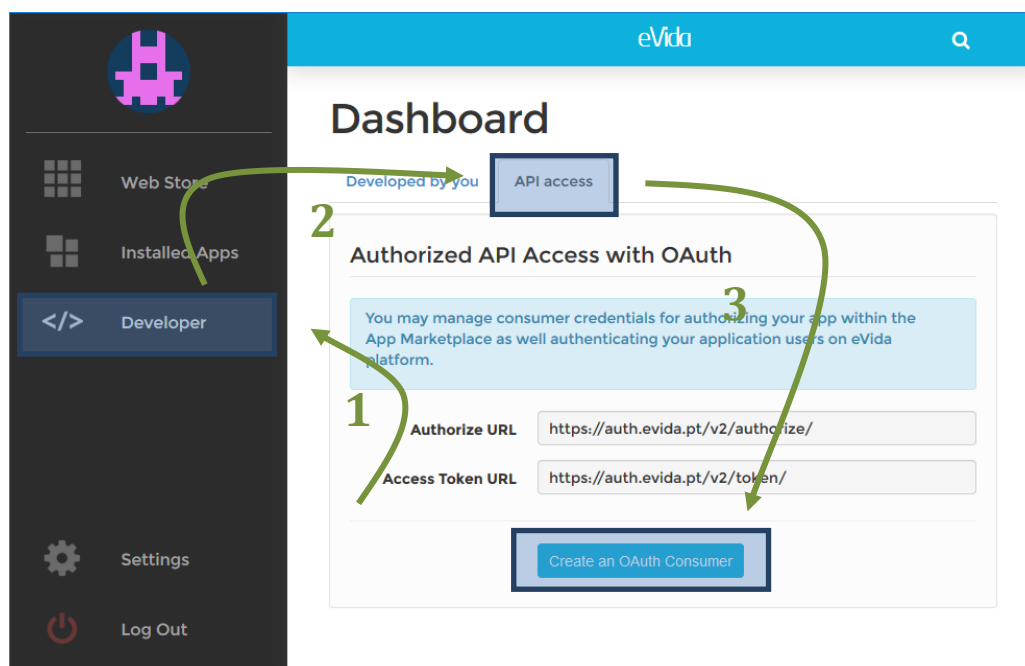
Figure 3: Create an OAuth Consumer

The **Consumer Key** and **Consumer Secret** are essential to obtain the access token (later described) which will allow full integration with eVida's API [1].

2.1.3 Activate the PHR application

Figure 4: Activate PHR application in eVida

To activate the PHR application (Figure 4), already available in eVida, one needs to first access it once.

The following steps describe the process to activate the PHR:

1. Login in eVida: https://www.evida.pt;

2. Locate the "Personal Health Record" application in the "Web Store" and click on it;
3. Click on "OPEN APPLICATION";
4. Allow application's access by clicking "Permitir acesso".

A graphical representation of these steps is shown in Figure 4

2.1.4 eVida's API

eVida platform provides a RESTful API [1] which allows getting and posting data between applications easy.

To demonstrate the interaction with APIs, here is used Postman, which is a tool for API developers (https://www.getpostman.com/). It allows an easy understanding and interaction with the REST APIs here presented.

2.1.4.1 Step 1: Login (getting the access token)

To get access to eVida's API first an *access token* has to be obtained. This is achieved through a POST request to https://auth.evida.pt/v2/token with the **Headers** and **Parameters** shown in Table 1.

POST	https://auth.evida.pt/v2/token
Headres	
Key	*Value*
Authorization	Basic Base64(<consumer_key>:<consumer_secret>)
Parameters	
Key	*Value*
client_id	<consumer_key>
grant_type	password
username	<evida_username>
password	<evida_password>
scope	user

Table 1: POST request to get an eVida access token. *Base64()* is a function to create a Base64 encoded string (e.g.: https://www.base64encode.org/).

Add this data to Postman and hit **'Send'**; the result should look like the figure below.
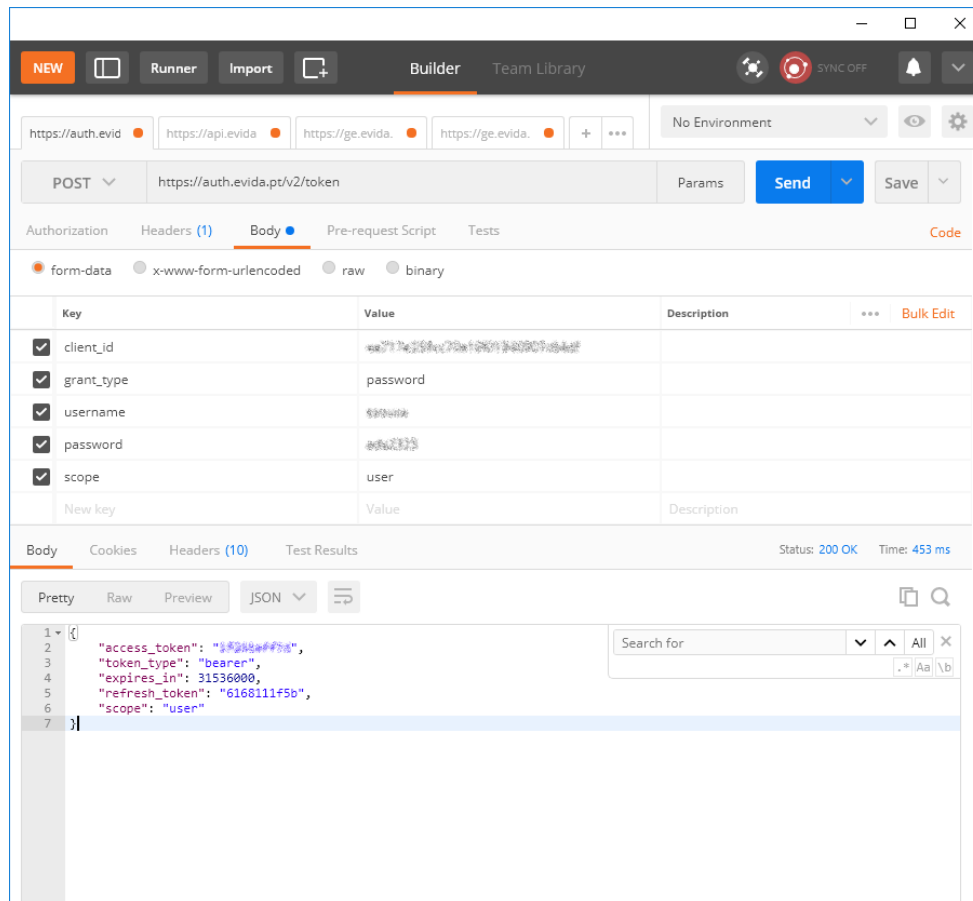
Figure 5: Postman POST response to https://auth.evida.pt/v2/token

The response will provide the `access_token`, which is the parameter to be used in all future eVida API calls.

2.1.4.2 Step 2: Getting the logged user

GET logged user in eVida:

GET	https://api.evida.pt/users?bearer_token=<access_token>

Table 2: GET request to retrieve logged user

The response will look like:
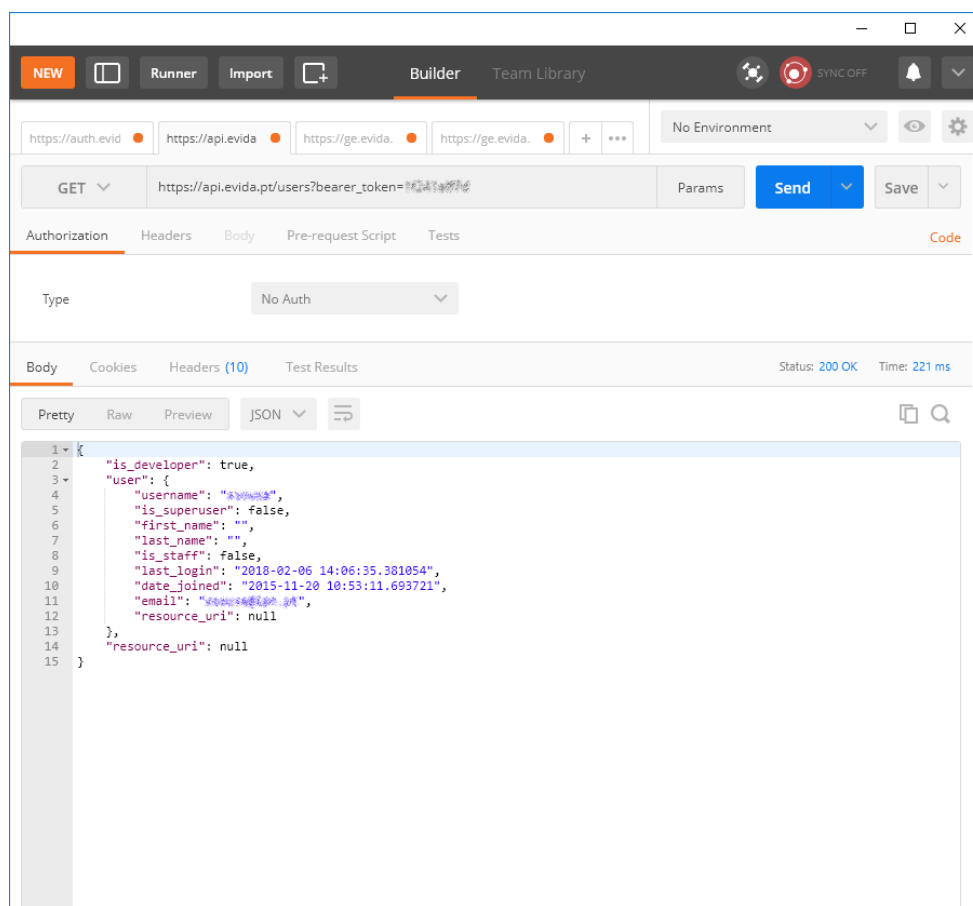
Figure 6: Postman GET response to https://api.evida.pt/users

This request retrieves some information about the logged user in eVida platform. The field "username" should be saved for further reference.

GET demographic information about the user:

GET	https://ge.evida.pt/PersonalProfile/items
Headres	
Key	*Value*
Authorization	Bearer <access_token>

Table 3: GET request to retrieve user demographic information
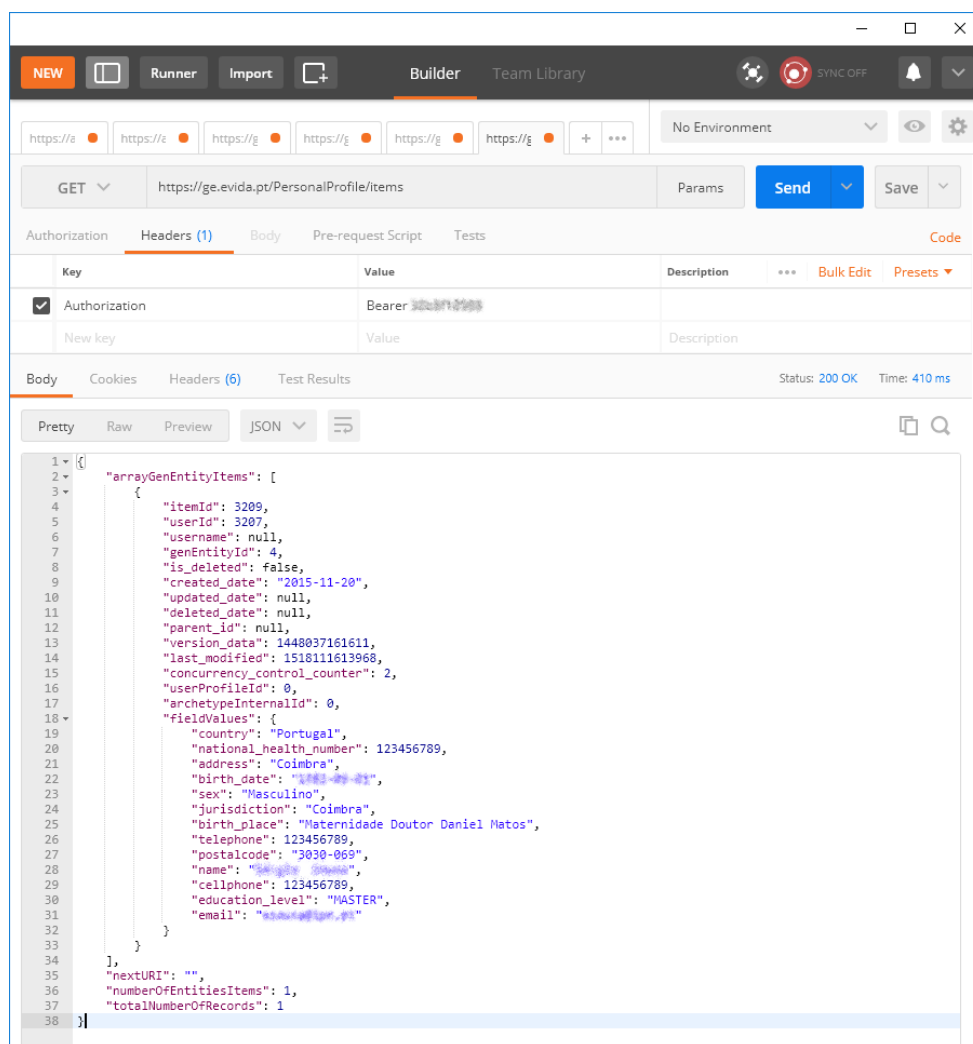
Figure 7: Postman GET response to https://ge.evida.pt/PersonalProfile/items

Getting list of shared profiles

GET	https://ge.evida.pt/otherUserProfile
Headres	
Key	*Value*
Authorization	Bearer <access_token>

Table 4: GET request to retrieve list of shared profiles
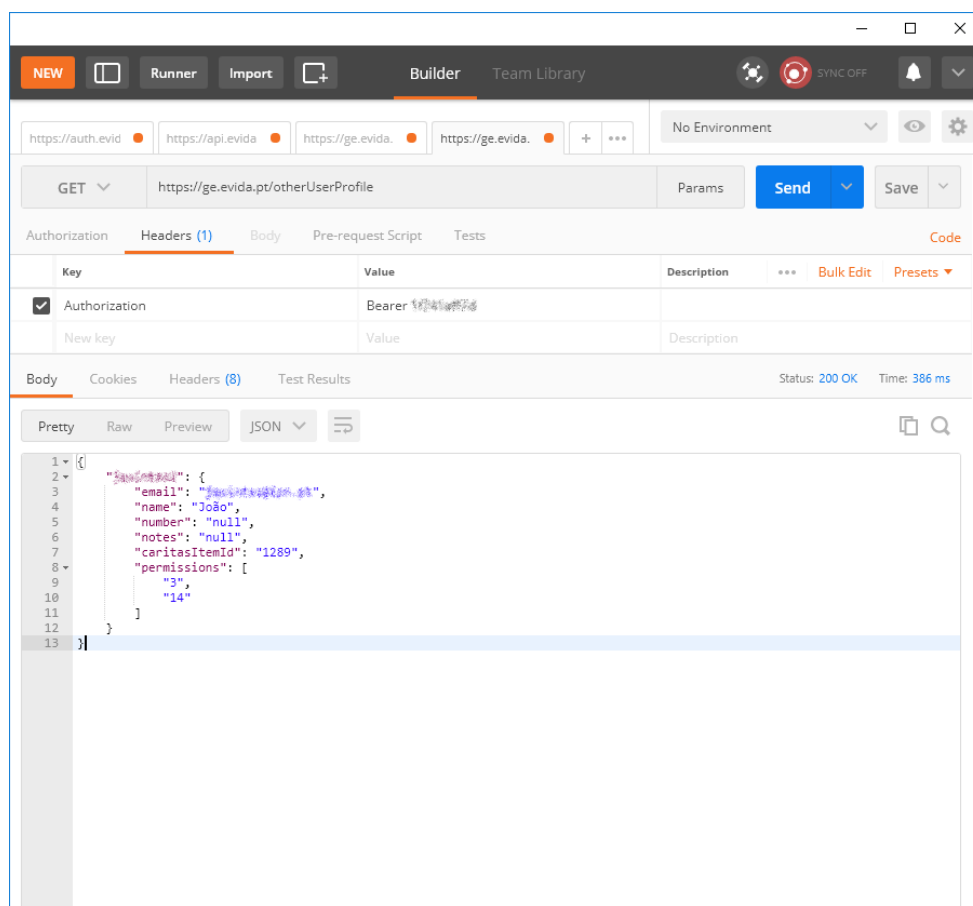
Figure 8: Postman GET response to https://ge.evida.pt/otherUserProfile

This request will return the list of users that have shared information with the logged user in PHR. This is very useful as one user can have access to multiple users' data information, allowing for one user being able to update others users medical information.

The data follows the structure:

```
{
      "username": {
            "email": ". . ."
            . . .
      },
      . . .
}
```

The important data to retrieve is the "username".

2.1.4.3 Step 3: Posting data to eVida (Archetypes)

eVida provides several Archetypes, where users can store sensor medical data in HL7 format. Below are presented the different archetypes supported at the moment.

Oximetry:

POST	https://ge.evida.pt/indirect_oximetry?access_token=<access_token>
Parameters	
Key	*Value*

apiRequest.userProfileUsername	<username>
measurement_date	15/01/2018
measurement_time	18:16:54
at0036	Nonin Onyx II 9560 Oximeter
at0006	97/100

Table 5: POST request to send data to the Oximetry architype. NOTE: *at0006* and *at0036* are fields for the Oximetry archetype, as defined in openEHR / HL7 v3 RIM
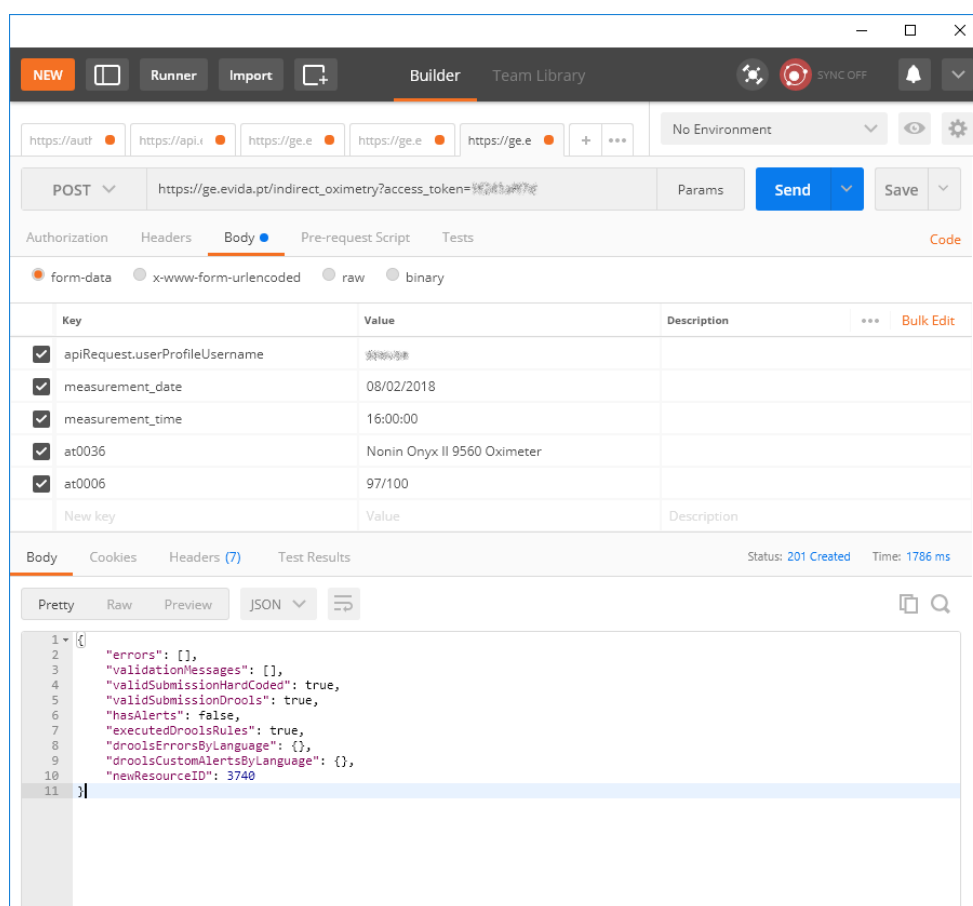
Figure 9: POST response to https://ge.evida.pt/indirect_oximetry

It is possible to post to a user other than the logged user. That is why having a list of the users that share the profile with the logged user is useful for this step.

Following are presented the other archetypes supported by eVida. The procedure to post data to those archetypes is similar to the method described above; only the data must be adjusted accordingly.

Pulse (Heart Rate):

https://ge.evida.pt/pulse

Figure 10: Pulse archetype

Body Weight:

https://ge.evida.pt/body_weight

Figure 11: Body weight archetype

Blood Pressure:

https://ge.evida.pt/blood_pressure

Figure 12: Blood pressure archetype

2.2 MySignals

MySignals SW includes three different modes to access all the information gathered from the sensors (Figure 13):

- **Standalone Mode** ([2], page 35): MySignals does not send the data. It is just visualized on the TFT screen.
- **Bluetooth Connection Mode** ([2], page. 48): MySignals send the data to a mobile phone using the BLE connection. The data is gathered and visualized by the MySignals App ([2], page 46). Then if the Cloud option is enabled, the data will be also transferred to the user's account at the Libelium Cloud.
- **Server Connection Mode** ([2], page 65): MySignals uses the WiFi connectivity to send the data directly to the Libelium Cloud.
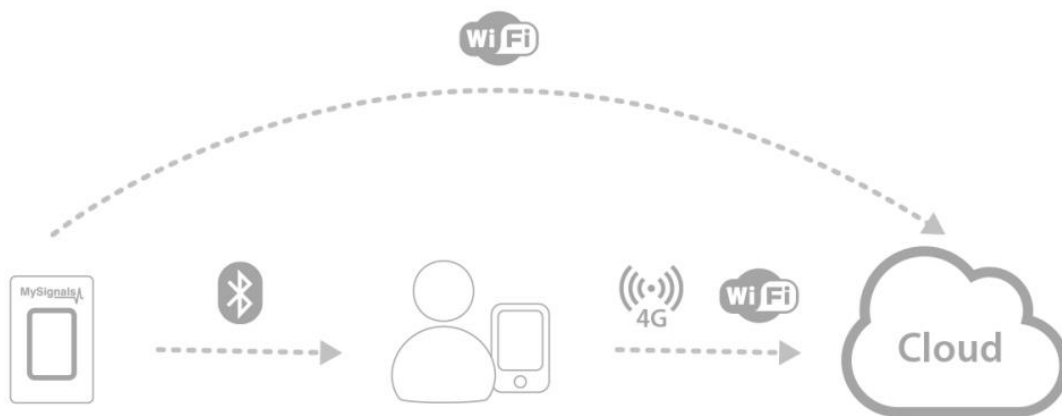
Figure 13: MySignals' accessing information modes

2.2.1 MySignals Account

This guide will describe briefly necessary steps that need to be accomplished in case any end user or developer wants to integrate any platform with MySignals Cloud. MySignals Web Server Application is a real-time large-dataset viewing and plotting tool and has built-in data analysis functionality. It is very user-friendly and contains many powerful built-in features. MySignals Web Server Application is an Application that allows you to configure MySignals for creating profiles and users and help you to visualize all the data measured. In order to access to the MySignals web application is necessary to have SCM account and a valid license. SCM stands for "Services Cloud Manager", and is Libelium's platform from where you can manage your devices a Cloud Services:

1. To create a free SCM account please fill this form and click in the validation email that you will receive:
   https://cloud.libelium.com/register
   (If additional help is needed please check the Services Cloud Manager Guide [3])
2. To activate your license you will need to follow this steps:
   - Obtain a license: in case you are a developer and want to create your own solution on the top of MySignals platform, you will have two different ways to obtain a license.
     o **Diatomic consortium has provided me one MySignals hardware device**: Libelium will give some free of charge MySignals devices to few developers/SMEs, so that they can test the technology and use it for their developments. In this case you will need to contact projects@libelium.com so that Libelium team can manage your request and provide to you a PRO developer license for free, valid within the lifetime of Diatomic project..
     o **I have decided to buy one MySignals hardware device**: in case you decide to acquire any of the MySignals kits available, you will also need to buy your MySignals license from Libelium's IoT Marketplace [4] or from your Sales agent.
   - You will receive in your email the license activation code.
   - You must click in the activation link that you will receive by email

After these steps you can start using MySignals Cloud service with the Website, Mobile APP and Developer API, in the terms and quotas contained in the license that you purchased.

As special arrangement for Diatomic project, in case you are not interested in the hardware, but you need some sample data in order to develop your own solution on the top of MySignals cloud, Libelium will enable a **test account** for developers so that you can test Libelium's API, and interact with it. Note that in this case, you will not receive any activation code and you will not be able to have access to all the functionalities of MySignals Cloud. In order to obtain a test account, you will need to contact projects@libelium.com. Libelium team can manage your request and provide you the credentials. This account will be valid within the lifetime of Diatomic project.

2.2.2 License activation

The "Licenses" section gives control of the licenses for the SCM. Licenses enable services for your devices. The "My Licenses" panel lists the currently active licenses and the historic data of all the expired licenses. If you recently purchased a License, go to the "Get Licenses" panel (Figure 14) to enter its activation code.

License activation codes are unique and only one use is allowed (one license only applies to one user, to one account). Despite the ownership of one device can be transferred (and one device can be managed by several users), the ownership of a license cannot be transferred.

Any license has an expiration time associated to it. It is important to note that time starts running from the moment the user activates it by entering its activation code (after entering the activation code of a license, it may be required to activate it in the "Service" panel).

If the user has one active license and activates a new one, time keeps running for both the old and the new licenses (in other words, time is never paused).

The "My Licenses" panel shows the licenses and the status of each one. The "Active" tab shows the list of the licenses which are currently active, and the "Expired" tab shows the list of previously used licenses.

Figure 14: MySignals "My Licences" panel

The information displayed is:

- License ID: Identification number
- Service: Service and type provided by the license
- Activated: Date of activation
- Expires: Date of expiration

License registration is done in the "Get Licenses" panel (Figure 15). It is a 2-step procedure: enter a valid license activation code (Figure 16) and confirm the action (Figure 17).
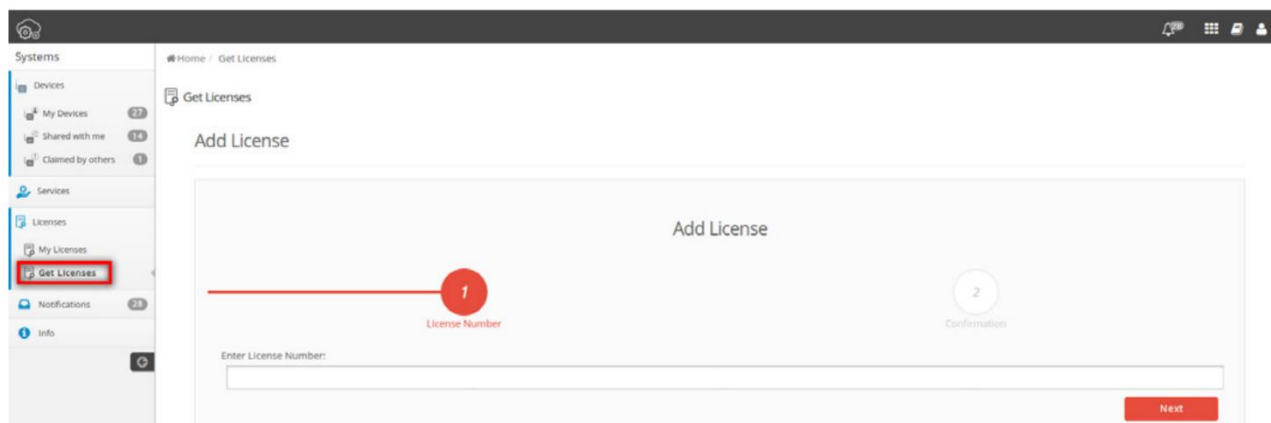
Figure 15: MySignals "Get Licences" panel

Entering a Single Activation Code will register one license. A Group Activation Code will register all devices belonging to the Sales Order, but not the licenses it may contain.
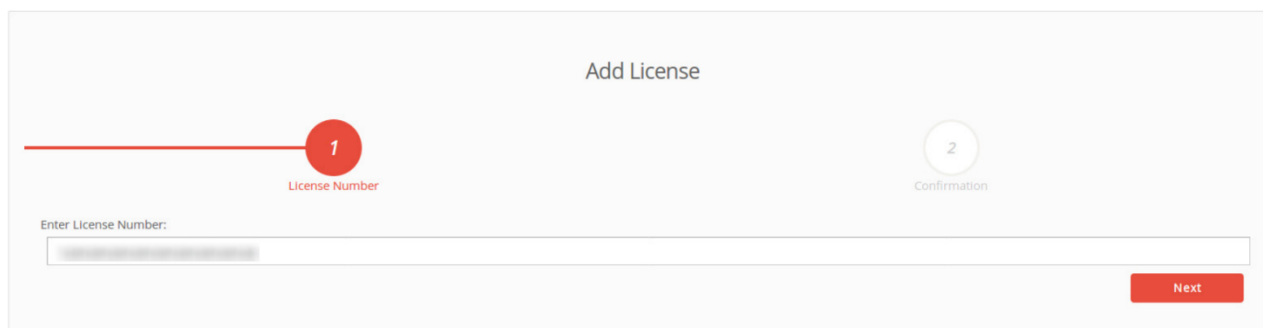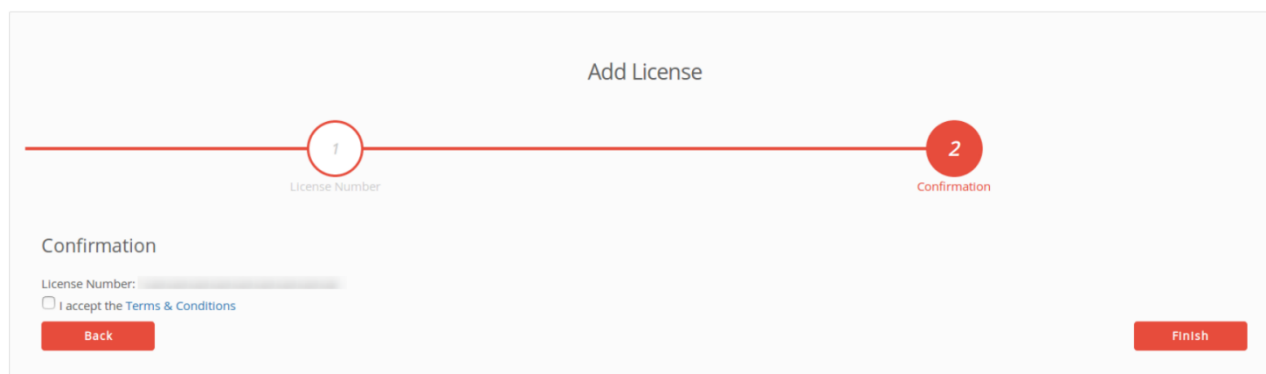

Figure 16: Add License Step 1

Figure 17: Add License Step 2

The SCM will validate the activation code, displaying a visual confirmation. When the process is finished successfully, a message will show that the license has been correctly added to the "My Licenses" panel (Figure 18).

Success!

License added

OK

Figure 18: License added successfully

If the process could not be finished correctly, a message will notify the error. For example, "invalid activation code: please check that the activation code you entered is valid" (Figure 19).

Something went wrong

License not valid

OK

Figure 19: License invalid

2.2.3 API Cloud

Developers may relocate the information stored in the Libelium Cloud to a third party Cloud server easily using the API Cloud provided.

2.2.3.1 A general overview

Libelium MySignals comes with a Cloud API that allows us to read data from our account. We can see a list our members and read the values measured for a user by MySignals. This data available in this RESTful API can be used by the customer to create new developments.

Libelium Cloud → www → [servers] → www → Third Party Cloud

There is no need to install anything but you can go to the representation of the API in Swagger format (Figure 20): https://cloud.libelium.com/mysignals_documentation/api_web/
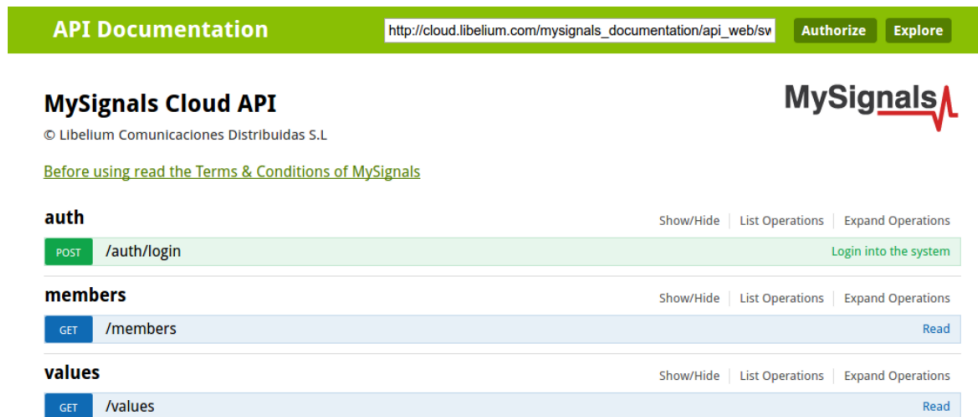
Figure 20: MySignals API in Swagger format

Here you can browse all the available methods of the API and see the parameters that you need to use. It is possible to test the API from here following steps

2.2.3.2 Step 1: Login (getting the access token)

Click over '/auth/login', fill the form with your email and password and click 'Try it out!'. If you provided the right data you should see something like this:
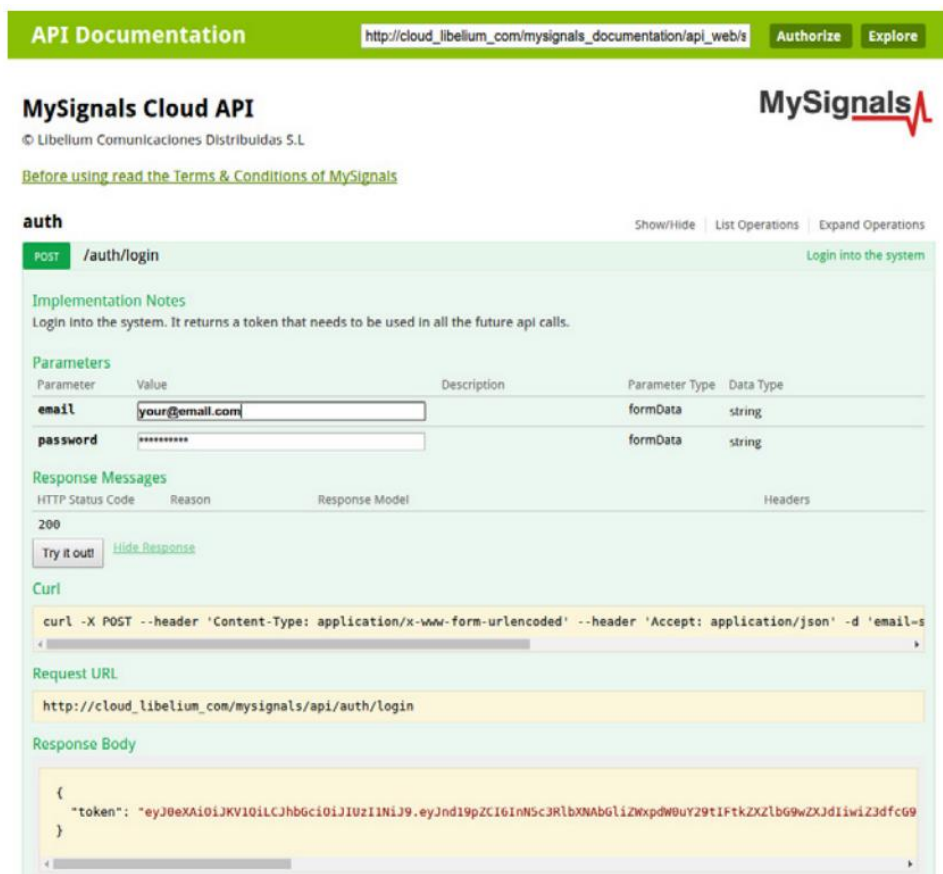
The response body contains the token that you should use to access to your data in the following steps. Click 'Authorize', write 'Bearer <your token>' and click authorize:

2.2.3.3 Step 2: Get list of your members

Click in '/members' section and then "Try it out!" button. You should see a list with your members. If you don't see it please make sure that you followed all the instructions of the previous step (Login).

With all the data that this method retrieves, you should have information enough to develop a webpage that could be similar to the one presented on the following image:

2.2.3.4   Step 3: Get the sensor values of a member

Click '/values' section and fill the parameters as in the picture. Then click "Try it out!" button. In this case you will need to introduce the sensor_id so that you can retrieve the desired information.

Sensor_ids available for retrieving data are:

sensor_id	name	units
position	Body position	1 supine, 2 left, 3 right, 4 prone, 5 stand or sit, 6 non-defined
position_x	X axis acc	g
position_y	Y axis acc	g
position_z	Z axis acc	g
temp	Temperature	º C
emg_cpm	Muscle contraction	cpm
ecg_bpm	Heart rate	bpm

airflow_ppm	Respiratory rate	ppm
gsr_us	Conductance	µs
gsr_ohms	Resistance	ohms
blood_dias	Diastolic pressure	mmHg
blood_syst	Systolic pressure	mmHg
blood_bpm	Heart rate	bpm
spo2_oxy	Oxygen saturation	%
spo2_bpm	Heart rate	bpm
gluco_mg	Glucose mg	mg/dl
gluco_mol	Glucose mmol	mmol/l
spir_pef	PEF	spir_pef
spir_fev	FEV1	spir_fev
snore_spm	Snore rate	spm
scale_ble_weight	Wheight	kg
scale_ble_bodyfat	Bodyfat	%
scale_ble_bonemass	Bonemass	%
scale_ble_musclemass	Musclemass	%
scale_ble_visceralfat	Visceralfat	%
scale_ble_water	Water	%
scale_ble_calories	Calories	kcal
blood_ble_dias	Diastolic pressure	mmHg
blood_ble_syst	Systolic pressure	mmHg
blood_ble_bpm	Heart rate	bpm
spo2_ble_oxy	Oxygen saturation	%
spo2_ble_bpm	Heart rate	bpm
gluco_ble_mg	Glucose	mg/dl
gluco_ble_mmol	Glucose mmol	mmol/l
eeg_ble_attention	EEG Attention	%
eeg_ble_meditation	EEG meditation	%
temp_ble	Temperature	ºC
button_ble	Alarm button	0 off, 1 on

2.2.3.5 Step 4: Last member data for all the sensors

The previous method requires specifying the sensor_id. Alternatively there is another method that allows to request all the last values registered for all the sensors. Click '/values/last_member_data/' section and fill the parameters as in the picture. Then click "Try it out!" button.

2.2.3.6 Step 5: Get a list of sensor raw data

Click '/raws' section and fill the parameters as in the picture. Then click "Try it out!" button. This method will answer with all the members that have linked any sensor data which wave signal can be represented graphically. You will need to keep the ID so that you can use it on next step.

Sensor_ids that have a wave signal representation available are:

Sensor_id	Name
airflow_raw	Airflow Wave Signal
ecg_raw	ECG Wave Signal
emg_raw	EMG Wave Signal
snore_raw	Snore Wave Signal

The method returns a list of the available raw data that can be requested in detail with the following method, as mentioned before.

2.2.3.7 Step 6: Get detail of sensor raw data

Click '/raws/{id}' section and fill the parameters as in the picture. Then click "Try it out!" button.

This method answers with all the necessary information for representing a wave signal graphically on your web page. You will be able to develop a tab in your dashboard that shows for example the ECG Wave Signal, as follows:
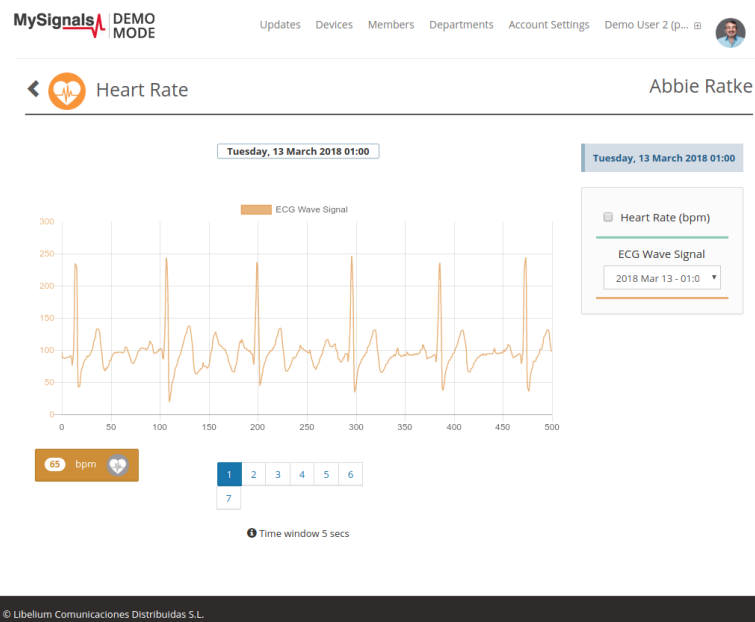
2.2.4 PHP example

There is an example (Figure 21) that you can download from:

http://downloads.libelium.com/mysignals/mysignals_web/api_cloud_v1.zip

Necessary steps:

- Extract the zip with the example
- Download the 'httpfull' library and place it in the /includes directory
  http://phphttpclient.com/downloads/httpful.phar
  Edit the file example.php and fill $email and $password with your values

- Go to your web browser and load the example.php page
- This will log you in the system, get a list of your members and get the latest 5 temperature values of one of your users

```
/*
 *
 * Copyright (C) 2016 Libelium Comunicaciones Distribuidas S.L.
 * http://www.libelium.com
 *
 * This program is distributed WITHOUT ANY WARRANTY; without
 * even the implied warranty of MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE.
 *
 * By using it you accept the MySignals Terms and Conditions.
 * You can find them at: http://libelium.com/legal
 *
 *
 * Version:          0.1
 * Design:           David Gascon
 */

include('includes/httpful.phar');

// Config
$email = 'your@email.com';
$password = 'your_password';

// API Vars
$api_base = 'https://api.libelium.com/MySignals';
$api_headers = ['Accept' => 'Application/x.webapi.v1+json'];


//1.- Login
$parameters = json_encode([
    'email' => $email,
    'password' => $password
]);
$response_login = \Httpful\Request::post($api_base . '/auth/login')
    ->sendsJson()
    ->body($parameters)
    ->addHeaders($api_headers)
    ->send();
echo "1.- Login: <br><br>".$response_login->raw_body."<hr><br>";

//Save the Token in the header array.
if($response_login->code == 200){
    $api_headers['Authorization'] = 'Bearer '.$response_login->body->token;
}
```

```
//2.- Get mymembers
$response_members = \Httpful\Request::get($api_base . '/members')
    ->addHeaders($api_headers)
    ->send();

echo "2.- Get my members: <br><br><pre>".json_encode($response_members->body, JSON_PRETTY_
PRINT)."</pre><hr><br>";




//3.- Get values from the first of my members
if(count($response_members->body->data) >= 1){
    $member_id = $response_members->body->data[0]->id;

    $parameters = [
        'member_id' => $member_id,
        'sensor_id' => 'temp',
        'ts_start' => '2015-01-01 00:00:00',
        'ts_end' => '2017-01-01 00:01:00',
        'limit' => '5',
        'cursor' => '0',
        'order' => 'desc'
    ];
    $response_values = \Httpful\Request::get($api_base . '/values?'.http_build_
query($parameters))
        ->addHeaders($api_headers)
        ->send();

    echo "3.- Get values from one member (member_id= ".$member_id."): <br><br><pre>".json_
encode($response_values->body, JSON_PRETTY_PRINT)."</pre><hr><br>";
```

Figure 21: PHP example code

2.3 Virtual Fort Knox Cloud Platform

Virtual Fort Knox (VFK) is a federative platform for the manufacturing industry developed by the Fraunhofer Institute for Manufacturing Engineering and Automation (Fraunhofer IPA). It will offer manufacturing companies an IT strategy that is cost-efficient, agile and scalable. Companies will be provided with efficient access to Industry 4.0 software solutions which are independent of manufacturers, in order to make advances in the digitalisation and optimisation of their production processes. Figure 22 depicts the concept of VFK. It is based on a cell structure and follows the "security-by-design" principle. Each VFK cell is a securely encapsulated environment for service users and service providers.
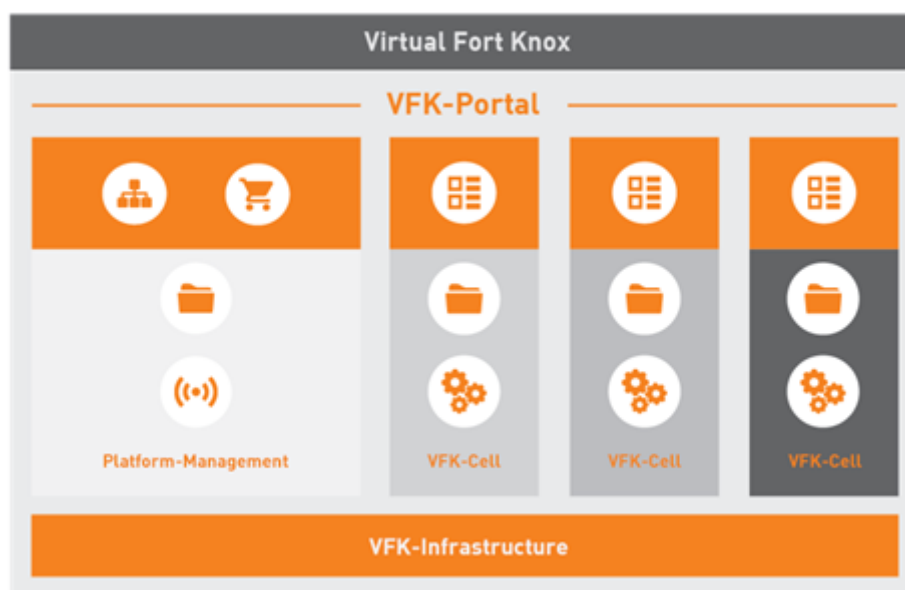

Figure 22: Virtual Fort Knox Concept

Figure 23 illustrates the VFK architecture. The physical devices are on the show floor level and are called Smart Objects (e.g. equipment or cyber-physical systems (CPS)). Due to the large number of communication protocols, a middleware is used for the communication. This middleware is called Manufacturing Services Bus (MSB) and is described in chapter 2.3.1. To communicate with IT services running in the cloud these services are also connected the MSB. Following a service oriented approach, the services can be aggregated to new services that provide new functionalities. Economically relevant will be the opportunity for Independent Service Vendors (ISVs) to offer their services in the VFK marketplace where the end users is be able to purchase the services they need. E.g. an equipment manufacturer can offer some special services for its equipment and the customers can purchase the services which they need. Form the technical side VFK offers a software development kit (SDK) which is available in all common programing languages. Applications can be hosted in the cloud infrastructure in form of virtual machines and docker containers. Additionally, the platform provides a flexible middleware as abstraction layer between components which allows changes to the flow of information at run-time.
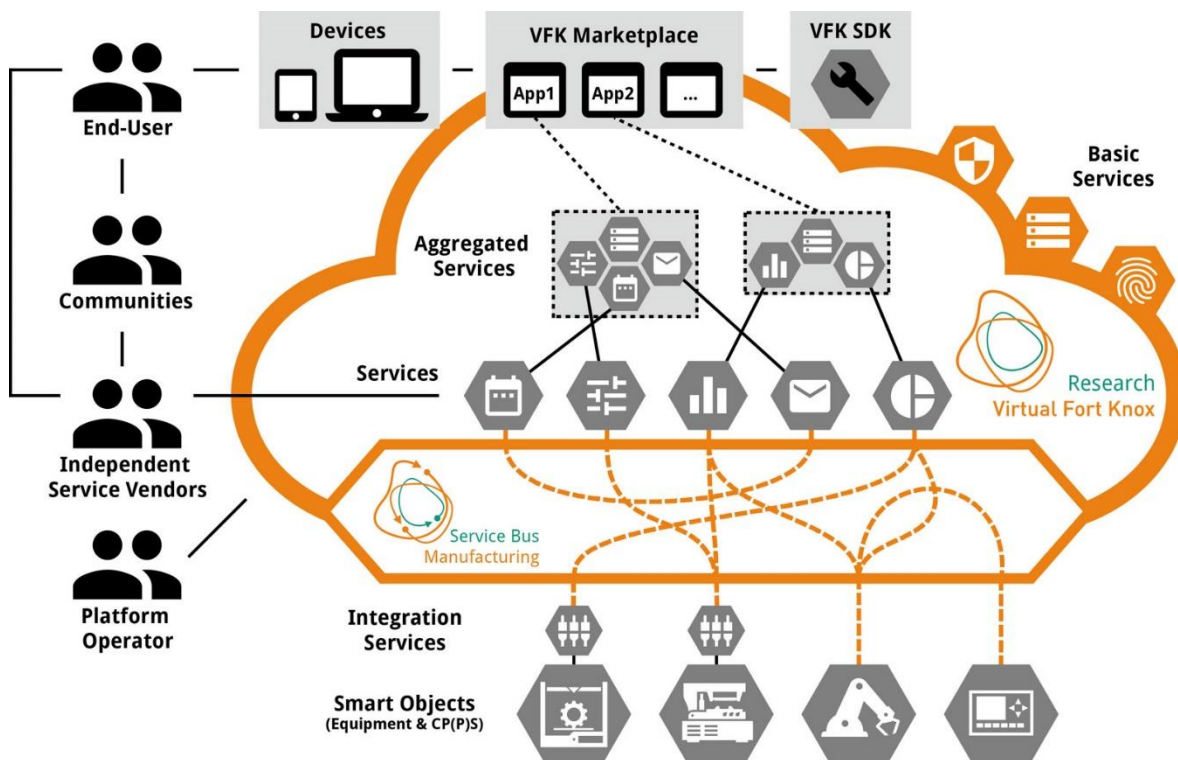

Figure 23: Virtual Fort Knox Architecture

2.3.1 Cell Concept of Virtual Fort Knox

Each organization, using VFK, operates within an encapsulated environment, referred to as a *cell*. These cells can be publicly hosted or run on a local machine in the network infrastructure of the organization, as shown in Figure 24. Data cannot be transferred between cells, unless applications or smart objects are specifically set up to do so (e.g. a bridge interface). Generally, it is advised to use the middleware accompanying VFK to set up communication within each cell. Each organization with its own cell may consist of multiple users. Users can deploy virtual machines or will be able to download preset software from the shop. Such preset software will be available in the centralized shop, which operates similar to other app-shops. By default, components which are deployed by a user are only visible, and therefore useable, to him. However, they can be made visible to other users within the organization as well.
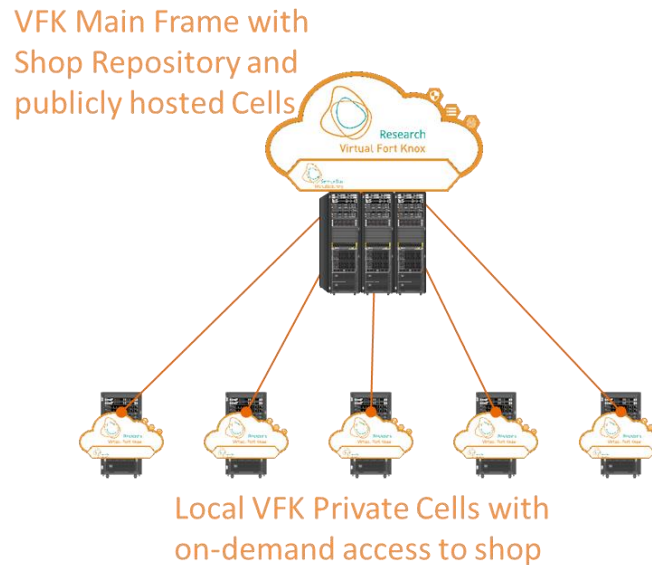
Figure 24: VFK cell concept with locally hosted cells and publicly hosted cells in the main infrastructure

2.3.2 Middleware - Manufacturing Service Bus

The Manufacturing Service Bus (MSB) enables a fast and low-effort integration of smart objects or IT-Services, because it provides the integration between various communication protocols such as RESTful Web Service or WebSocket API and various communication standards, for instance OPC UA. For this purposes the MSB provides common interfaces which allow the communication between smart objects and IT-services. The communication process is shown in Figure 25. The data are transferred in an encrypted channel. All send data are transformed to a common data format which ensures that all communication participants can communicate with each other. Received data are added to a queue to allow communication between communication partners with different communication cycles. The routing of the data is done using so-called integration flows, which allow the users to flexibly define where data is forwarded to. Integration flows can be defined without programming skills in the web based user interface of the MSB. Alternatively, a RESTful API is also available for automation purposes.

Figure 25: Communication Process of the Manufacturing Service Bus

2.3.2.1 Communication pattern

The communication follows the pattern depicted in Figure 26. At the start of the communication, the client registers itself with the appropriate interface (depending on the used communication protocol). When registering, the client sends its self-description, so that the MSB knows who is registering and what capabilities are available and which data can be expected. After registration is done the client can send data by throwing an event that contains the data. To send data to the client the MSB calls the appropriate function on the client with the data as function parameters.
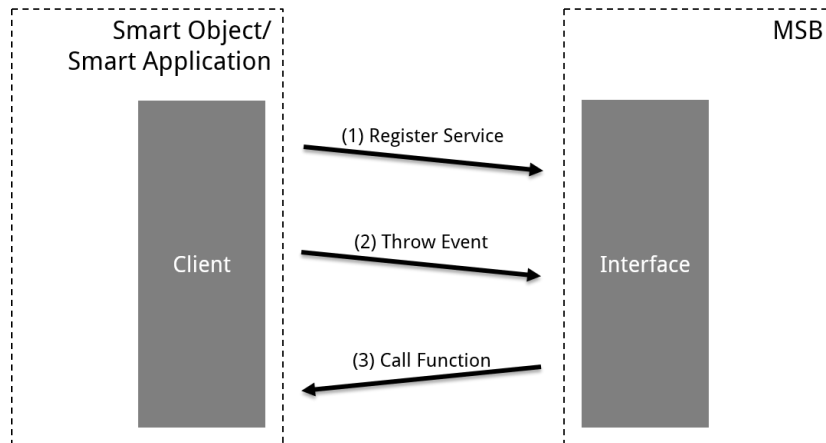

Figure 26: Communication Pattern of the Manufacturing Services Bus

2.3.2.2 Self-description of services

Each service has a self-description describing its characteristics. The structure of the self-description is shown in Figure 27. A service is classified as an Application or as a Smart Object and can be identified by its unique UUID. Data that is send by a service is described as events. Data that is send to the service can be received as function. Functions can be used to trigger capabilities of the service by internally mapping the incoming function to a callback function in the service-specific code. Such a callback function can trigger return events as well.
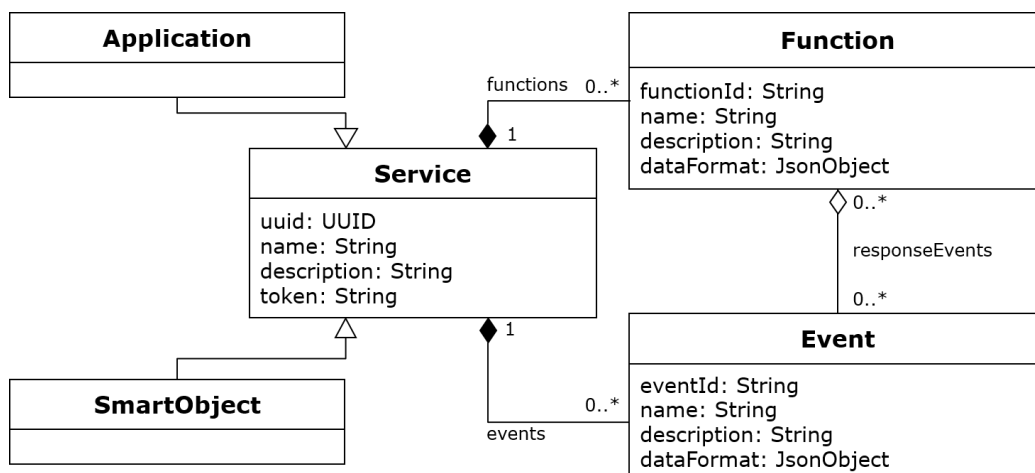

Figure 27: Self-description of services

Once a component (smart object or application) is registered at the MSB, the MSB can be configured to transfer information to and from the component automatically by manual configuration via a graphical user interface (GUI). A simple example for the communication pattern is shown in Figure 43. To achieve the shown information exchange, three main configuration steps have to be completed: selection of the two components, selection of the corresponding event and function and finally mapping of output data of the smart object to the input parameter of the function of the application. The data emitted by the smart object is attached to the event as a JSON string. The data is then mapped to the corresponding input parameters of the function of the smart application and wrapped in a JSON string again.
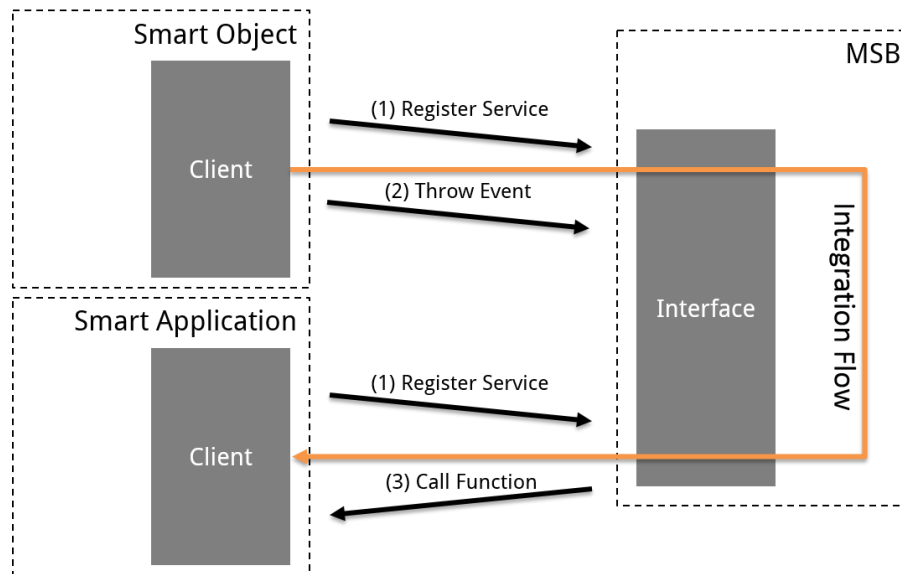
Figure 28: Exemplary pattern for data transfer of smart object to an application

For websocket communication ready-to-use client libraries in several programming languages are available, which developers of smart objects and applications can use to connect their own product to the MSB.

2.3.2.3 Meta data format

The data format of events and functions is shown in Table 7. It is based on the OpenAPI Specification 2.0 (aka Swagger Specification) derived from the JSON Schema for programming language independent definitions of data format. The complete OpenAPI specification that is used for the Swagger-UI as well as for the applications JSON definition can be found under https://github.com/OAI/OpenAPI-Specification.

Listing 1 shows an example for the specification of a complex object.

Table 6: Meta data format of the Manufacturing Service Bus

Common Name	Type	Format	Comments
Integer	integer	int32	signed 32 bits
Long	integer	int64	signed 64 bits
Float	number	Float	
Double	number	double	
String, Short	string		
Byte	string	Byte	
Boolean	boolean		
Date	string	date-time	As defined by date-time - RFC3339

Common Name	Type	Items	Comments
Array, List, Set	array	\<items\>	

Common Name	Type	Properties	Comments
Model	object	<properties>	

Common Name	$ref		Comments
Reference	#/definitions/<Model>		

Listing 1: Sample of a complex object specification

```
{  "dataObject":{
     "$ref":"#/definitions/alarm" },
   "alarm":{
     "type":"object","properties":{
       "reason":{"type":"string"},
       "errorCode":{"type":"integer","format":"int32"},
       "machine":{"$ref":"#/definitions/machine"} } },
   "machine":{
     "type":"object","properties":{
       "id":{"type":"integer","format":"int64"},
       "name":{"type":"string"}
     } } }
```

2.3.3 RESTful API

2.3.3.1 Registration

The MSB supports plain old REST to communicate with applications and smart objects. While the WebSocket interfaces allow to infer the connection state of connected smart objects and applications, the REST interface does not allow this. The reason for this is the stateless nature of REST interfaces.

The MSB's REST interface can be reached at port 8083, regardless of the cell the MSB resides in. The OpenAPI specification can be found at the same port under the path /swagger-ui.html. That means, that for MSB reachable under the URL msb.vfk.de, the REST API would be reachable under the URL msb.vfk.de:8083, with the API documentation available under msb.vfk.de:8083/swagger-ui.html.

The Swagger-UI is a documentation tool for APIs that provide an OpenAPI specification. The MSB provides such a specification. While the OpenAPI specification should provide enough information on the interface for basic use cases, it does not provide enough information for complex applications. This documentation serves to supplement the Swagger-UI specification.

As shown in Figure 29, the self-description described in Figure 27 has been extended to support the integration of the endpoints of a RESTful application. A RESTful application can be registered to the MSB in two different ways. One possibility is to use the MSB GUI and the other one the REST API of the MSB.

2.3.3.1.1 Registration with REST API

The REST API two endpoints: One for the registration of an application (Figure 30) and the other one for the registration of SmartObjects (Figure 31). The self-description of the application that should be registered must be described as a JSON object. Listing 2 shows such a JSON description of a simple application and is meant to serve as an example. The fields contained in the JSON definition are described in Table 7.
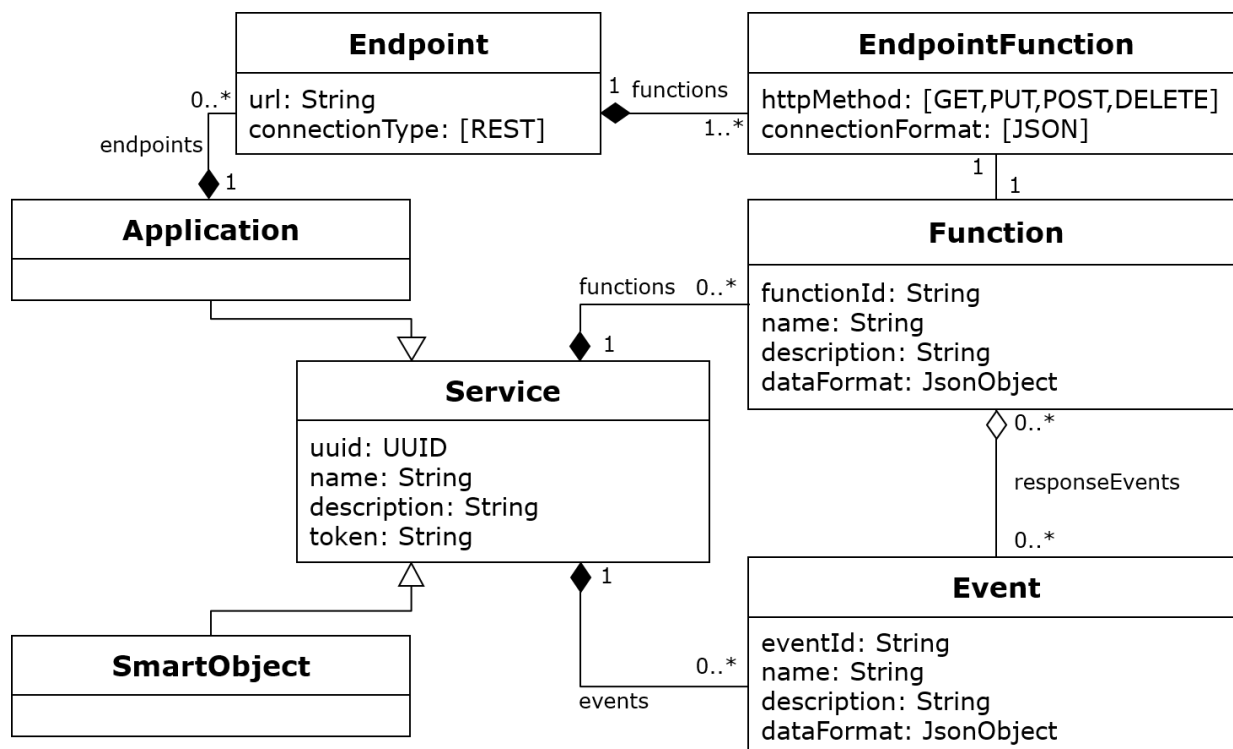
Figure 29: Extended self-description for RESTful API

Figure 30: REST endpoint to register application

Figure 31: REST endpoint to register SmartObject

Listing 2: Sample self-description of REST application as JSON object

```
{ "@class": "Application",
  "uuid": "71f747a8-b12e-476a-bdb7-85c68c59c282",
  "name": "System Information",
  "description": "Provides information about a remote system",
  "token":"auniquestring",
  "events": [ {
    "@id": 1,
 "dataFormat": {
   "dataObject": {
     "type": "object",
       "properties": {
         "system": {"type": "string"},
         "name": {"type": "string"} } } },
"description": "Displays live information about a remote system.",
"eventId": "live-information",
"name": "Live System Information" }],
 "functions": [ {
    "@id": 1,
"functionId": "/system-information",
"name": "System Information",
"description": "Provides System information",
  "dataFormat": {},
    "responseEvents": [1] }],
 "endpoints": [ {
     "url": "http://www.example.com",
     "connectionType": "REST",
     "functions": [ {
        "httpMethod": "POST",
        "connectionFormat": "JSON",
```

```
        "function": 1 } ]
  } ]
}
```

Table 7: Description of fields contained in JSON object of a self-description

@class	Can either be Application or SmartObject, depending on which type of object is described.
uuid	A unique identifier for the application. A valid identifier can be generated under https://www.uuidgenerator.net/version4.
name	The name of the application that will be displayed on the MSB GUI.
description	A textual description of the application
token	A token that will be entered in the MSB GUI to complete the registration of the application
events	A JSON description of the events that the application provides to the MSB. The defined events have to provide in incremental numerical id, as well as a unique textual event id. The numerical id is used to refer to the event as a response event in a function definition, while the textual id is used to route the events information within integration flows. The dataFormat follows the OpenAPI specification 2.0. The outer object of the data format must be called dataObject and must be of type object. Everything within the dataObject is optional and can be defined by the developer.
functions	A JSON description of the functions that the application provides to the MSB. The defined functions have to provide in incremental numerical id, as well as a unique textual function id. The numerical id is used to refer to the event as a response event in a function definition, while the textual id is used to route the function information within integration flows. Additionaly, the functionId is the path that is attached to an endpoint. If a function can be reached under the url www.example.com/someFunction, the functionId must be /someFunction. The numerical ID is used under endpoints to connect endpoint definitions with functions. The dataFormat follows the OpenAPI specification 2.0 and is completely developer defined.
endpoints	A description of the endpoints under which the application can be accessed by the MSB. A URL and a connection type have to be provided. The functions section further describes how the functions can be accessed by the MSB. The function attribute in the specification refers to the @id attribute of a function defined in the outer application description scope.

2.3.3.1.2 Registration with MSB GUI

A REST Application can be also added using the MSB GUI. Therefore, you must select the APPLICATIONS tab and press the "+" Button in the left corner. In the pop up you must press "Create App" and the "Manual app creation wizard" as shown in Figure 32 will appear. In Step 1 an UUID is automatically generated and you can enter other basic information like the name and the description of the application.

Figure 32: Manual app creation wizard – Step 1: Basic Information

In Step 2 (see Figure 33) the URL of the REST endpoints must be defined.

Figure 33: Manual app creation wizard – Step 2: Endpoints

In Step 3 (see Figure 34) for each defined REST endpoint functions can be defined. The path can contain parameters in the form of {parameter1}. This data format of the parameters must be defined in the "Request Schema" as described in section 2.3.2.3. If the function is called via an integration flow the parameter can be mapped from the triggering event. The parameter will be replaced with the value of provided by the event and the REST call will be executed.

Figure 34: Manual app creation wizard – Step 3: Functions

The response of the executed REST call will be send as a response event. The response events for the functions can be defined in Step 4. In the "Response Event Schema" you must described the data format of the data that will be responded by the REST application (see section 2.3.2.3).

Figure 35: Manual app creation wizard – Step 4: Response Events

In the last two steps you can verify your input and finish the wizard. After that the application will automatically appear in the applications list.

2.3.3.2 Send data

Once your application has been registered and verified to the MSB it is ready to receive information via a function call and to send information to the MSB via an event. An event is send as JSON object with the fields described in Table 6.

Table 8: Description of fields contained in JSON object of an event

uuid	UUID of the Service that sends the event.
eventId	Id of the event as definied in the self-description of the Service.
priority	Priority with which the event is to be processed by the MSB
dataObject	JSON object that contains the data of the event.

The event is then sent to the REST endpoint (/rest/data) shown in Figure 36.

Figure 36: REST endpoint to send data

2.3.3.3 Registration and verification of a component

Upon first boot up of a service, the service connects to the MSB and provides a self-description (more on how to achieve this and details on the self-description in section 2.3.2.2). At this point the component is not yet visible to the user, who needs to activate it first. To do so the user has to navigate to the *SMART OBJECTS* or *APPLICATION* tab, depending on the self-description of the component and click the button for new components ("+"), as can be seen in Figure 37. The classification into smart objects or application is based on the self-description, set by the developer. Its purpose is to allow easier distinction for human users and has no further implication beyond that. As a rule of thumb, smart object should contain at least one sensor or actuator.
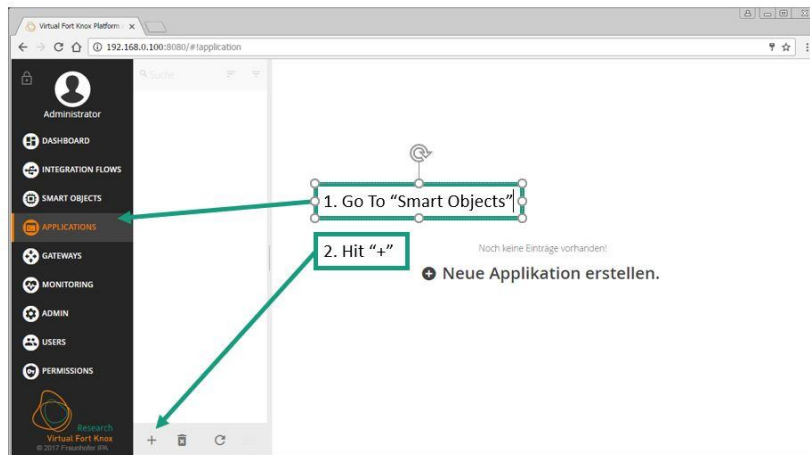
Figure 37: Activate new Component in GUI of MSB

The user is then asked to verify the component by inserting the security token which is part of the self-description and should be provided by the developer. At this stage the user can also decide if the component should only be visible to him or the entire organization (see Figure 38 below). Visibility decides if other user can see the component when they connect to the MSB with their user credentials. If they do not see the component, they cannot set up new information flows including the component. However, they still might be affected, as information flows set up by one user by trigger actions in organization wide visible components. An example might be a component which takes a considerable amount of time to process data and blocks any other request in this state. One user may find the component permanently blocked when setting up his information flow without any capability to identify who is blocking the component, since he cannot see the integration flow of another component which is invisible to him.
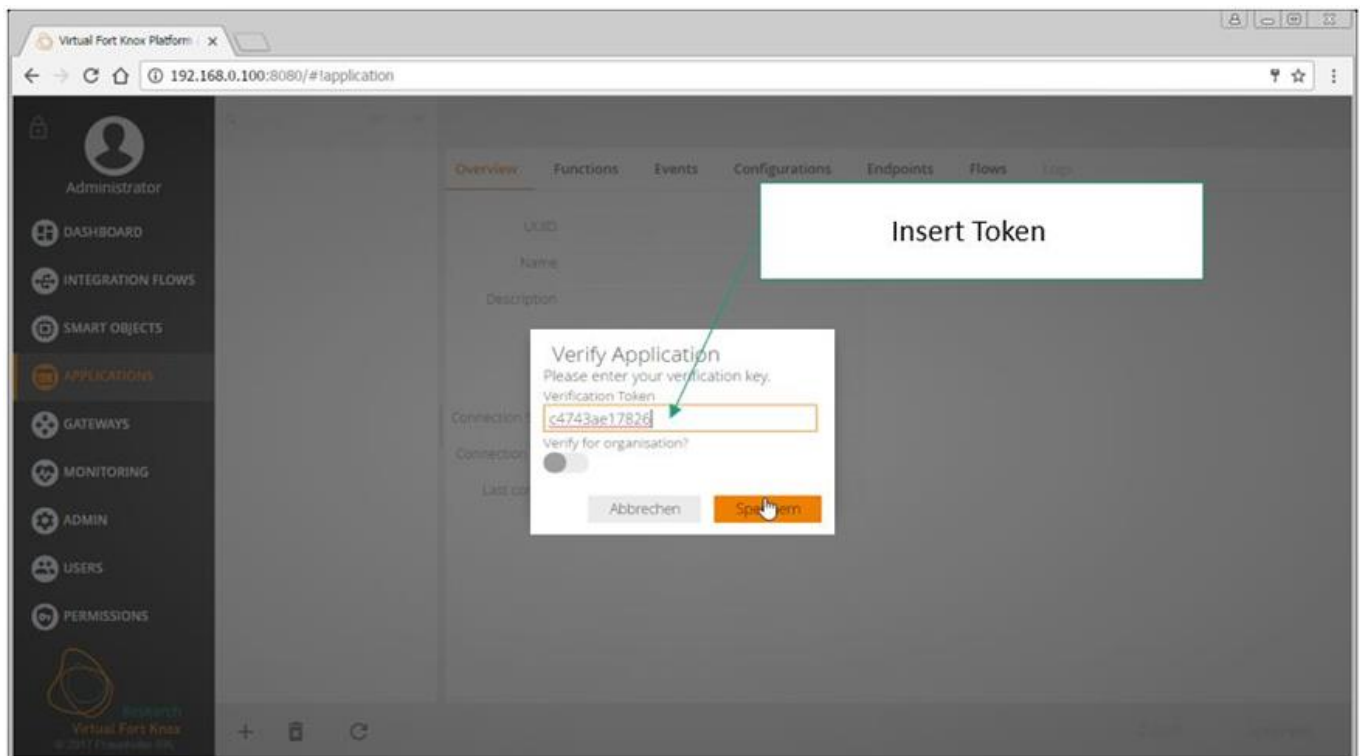

Figure 38: Security Token and Visibility Setting (MSB GUI)

2.3.3.4 Reviewing of registered components

After completion of the registration process, the new component will be listed in the appropriate tab (*SMART OBJECTS* or *APPLICATIONS)* with its self-description. This description includes general information like name and prose description of its general purpose, as well as specific information regarding the outgoing events the component can throw and functions which can be linked to incoming events, as seen in Figure 39. The component is now ready for the modelling of information flows. If the user wishes to delete the component, he can do so at any time by selecting the component in the respective tab, clicking the small garbage bin icon in the bottom left and confirming his decision.
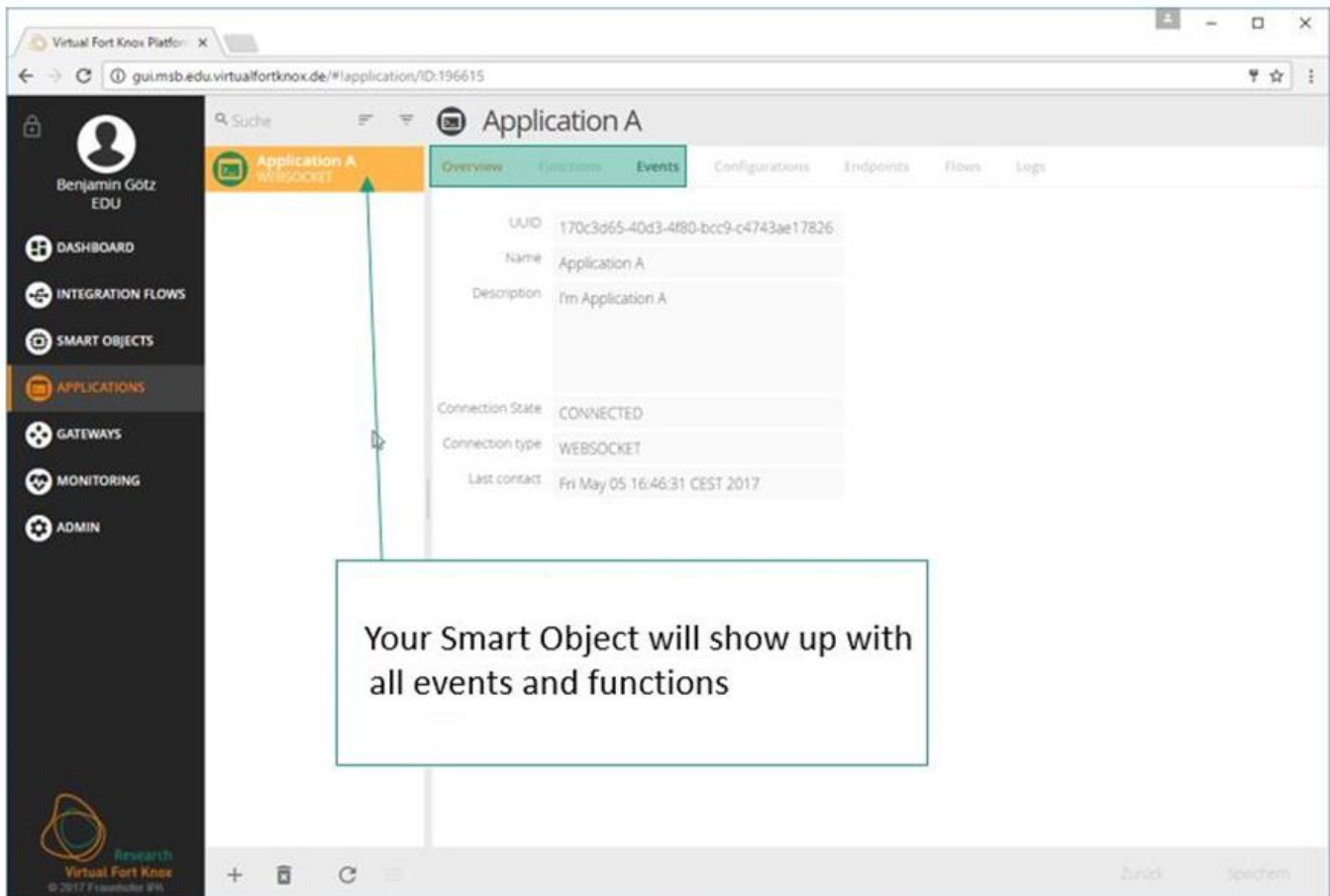
Figure 39: Detailed Information about Component (MSB GUI)

2.3.3.5 Settings configuration for components via MSB

Optionally, the *configurations* tab, shown in Figure 40, allows the user to configure internal values of a component remotely via the MSB. This feature is optional and has to be set up by the developer in the program code of the component. If no remote configuration is allowed, the tab name is greyed out and inactive. New parameters can be set by change the value on the right and pushing the orange save button on bottom right. If the new value does not correspond to the required format indicated in the middle row, the changes will not take effect. If a component is modified which is currently offline, the changes will not take effect.

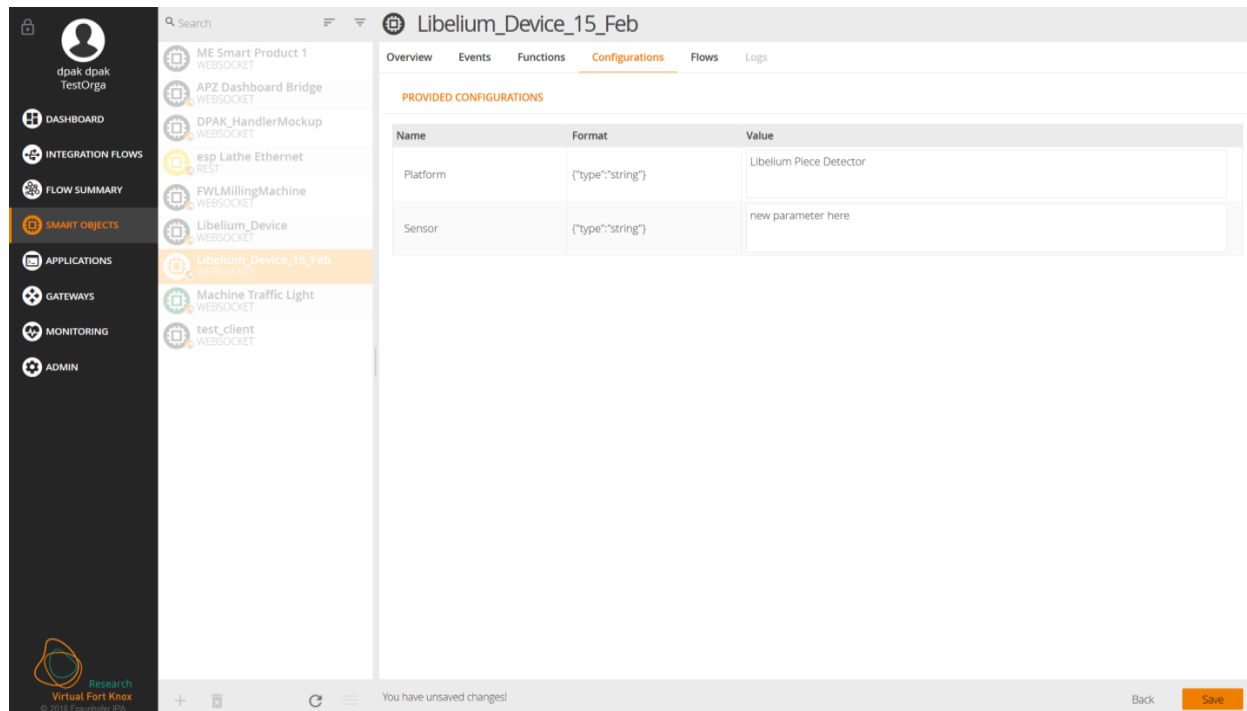**Figure 40: Detailed View on Component - Configurations Tab**

The *Flows* tab show all integration flows which the component is currently a member of (see Figure 41). When activating a new component, this tab is empty. Clicking on the arrow on the right lets the user jump directly to the selected flow.
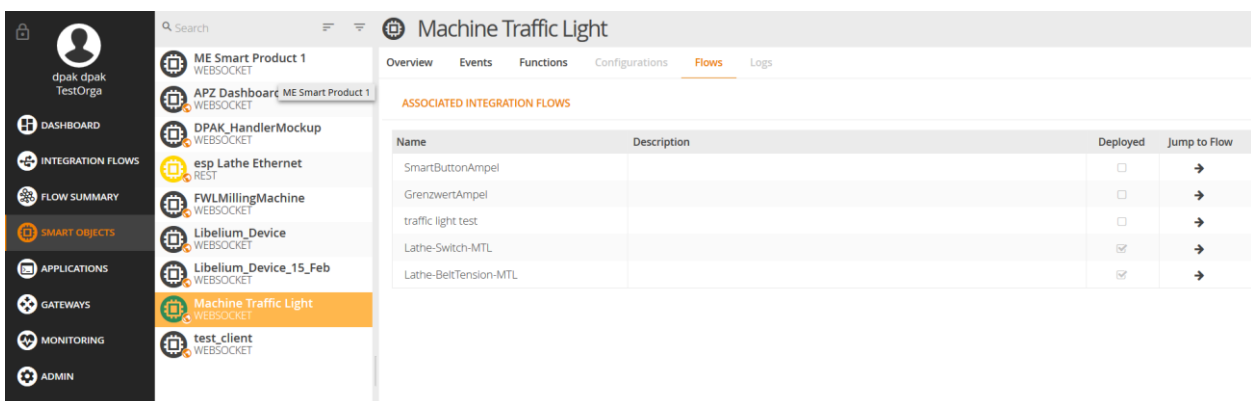

Figure 41: Detailed View on Component - List of all associated Flows (MSB GUI)

2.3.3.6 Data routing with integration flows

2.3.3.6.1 General information about integration flows

On the one hand, the middleware approach replaces the otherwise required point-to-point connections between the components and reduce the maintenance effort for the IT-personnel. On the other hand, it takes over the function of the event listener for all components and allows the configuration of this function at run time. This simplifies the adaption process in case changes to existing solutions are required (e.g. replacement of an old component). It also simplifies the implementation of new solutions, which rely on data or processing capabilities of existing components and can be configured at run-time without shutdown of the entire system.

To understand how to design components for the use with the MSB, it is useful to first understand, how the user configures his solution based on the available components. Figure 42 shows a simple example for such a solution from view of a user who configures two components to communicate. Once the

integration process is complete, the left component can send information to the second one for further processing. The user who implements the solution used a building block concept in a graphical user interface to setup this connection. To achieve the desired information flow, three conditions need to be met, which are represented in Figure 43:

- The components are registered with the MSB (self-description is provided) and activated.
- The information flow is modelled in the MSB by the user.
- The information flow is triggered by the first component in the chain at run-time.
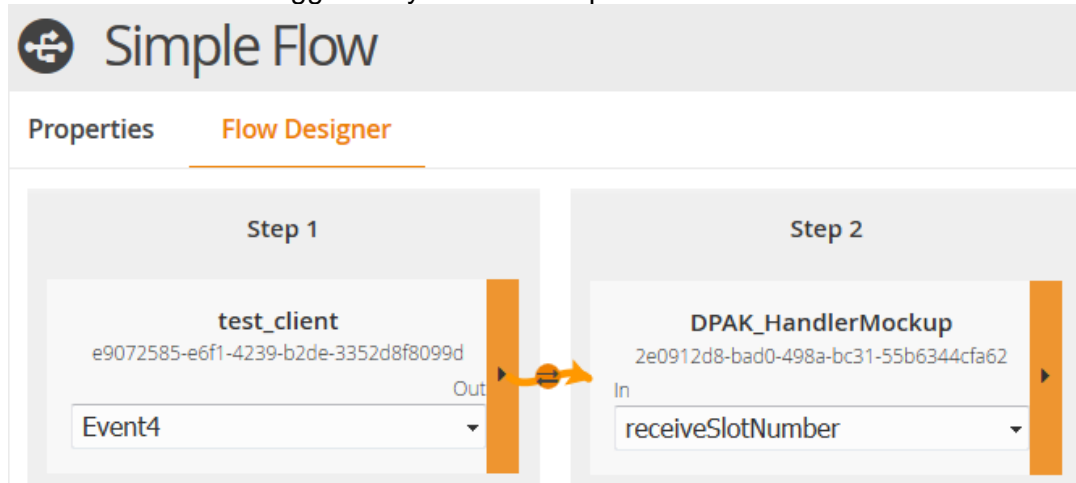
Figure 42: Simple Information Flow modelled in the MSB GUI (only part of GUI is shown)
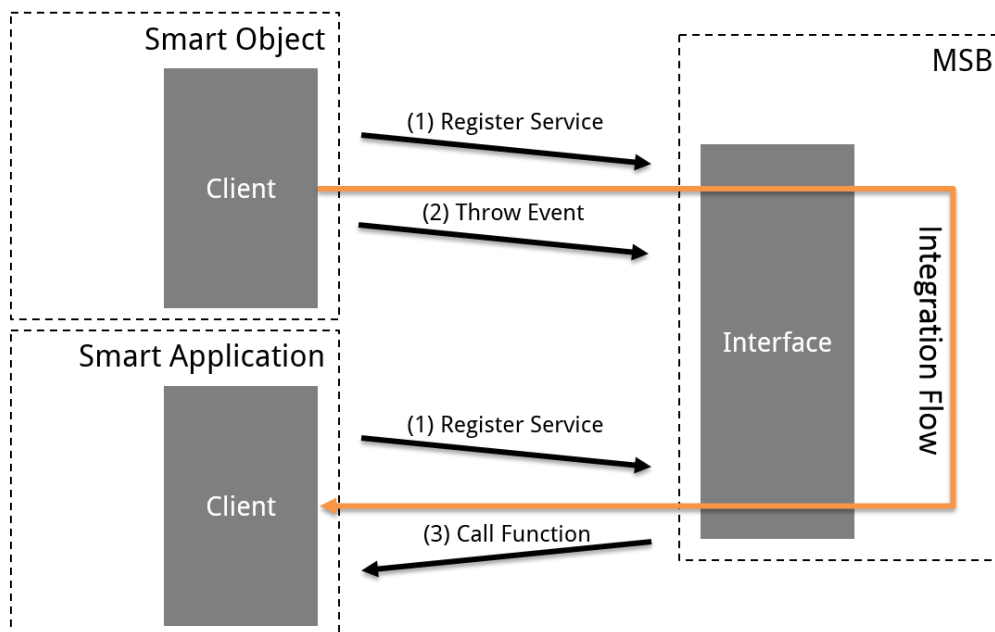
Figure 43: Steps to compete the Exchange of Information

It is important to understand that each component solely communicates with the MSB and does not per se know preceding or subsequent component in the chain of information. This has implications on the design as intrinsic knowhow between components cannot be expected and should therefore be avoided and all contextually required information needs to be available in the self-description or needs to be exchanged in the events.

2.3.3.6.2 Modelling of an Information / Integration Flow

The MSB is technically able to map events from a component onto functions of the same component. However, this capability should not be used in general to keep the load on the MSB low. Besides, the latency of the MSB typically exceeds component-internal communication by a large margin due to the underlying IP-based communication.

2.3.3.6.3 Initial Creation of an Integration Flow

To build a meaningful information flow, at least two separate components are required. In context of the MSB a model for an information flow are called *integration flow*. Figure 44 shows the first step in creation of such a flow in the *INTEGRATION FLOWS* tab which is initiated similarly to the activation of a component. Once the blank flow is created, it needs to be named, while a description by the user is optional. The modelling can then be initiated by clicking the *Flow Designer* tab (3. in Figure 44).
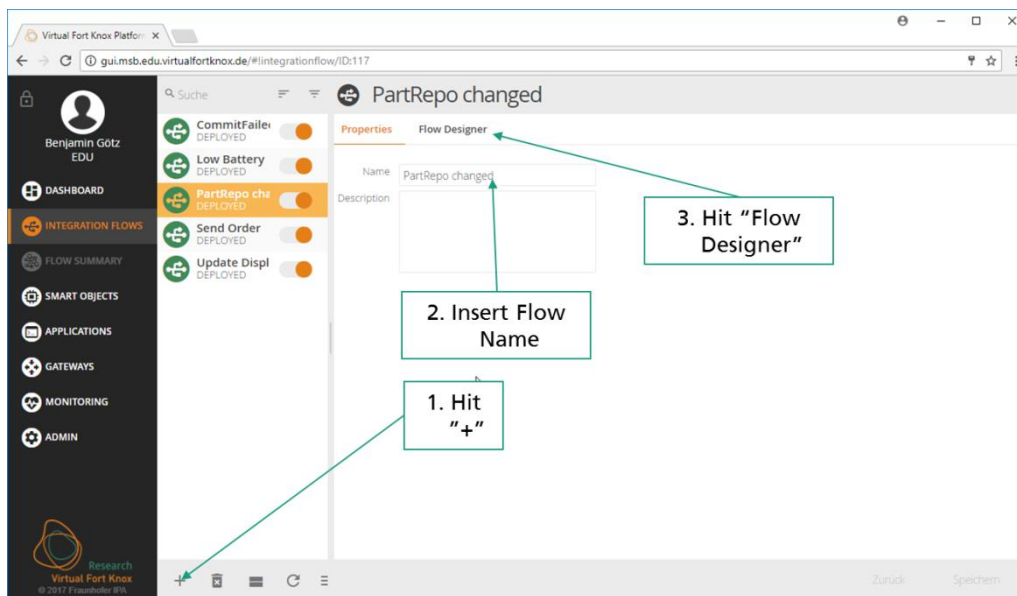

Figure 44: First Step in Creating a new Integration Flow (MSB GUI)

2.3.3.6.4 Selection of Components for an Integration Flow

Within the flow designer view, all available components are show on the right side. If the desired component is not shown, the list can be extended to show all smart objects or all applications by clicking on the respective fields. If the component cannot be found, it has not been activated and the steps of section 2.3.3.3 need to be competed again. The components, required for the integration flow need to be dragged and dropped onto the main area, as indicated in Figure 45.

After dragging all components into the main area of the GUI, the user should check if all components are positioned in the correct order according to the desired information flow from left (first component in chain) to right (last component in the chain). This is not necessary but advised, as it improves readability.
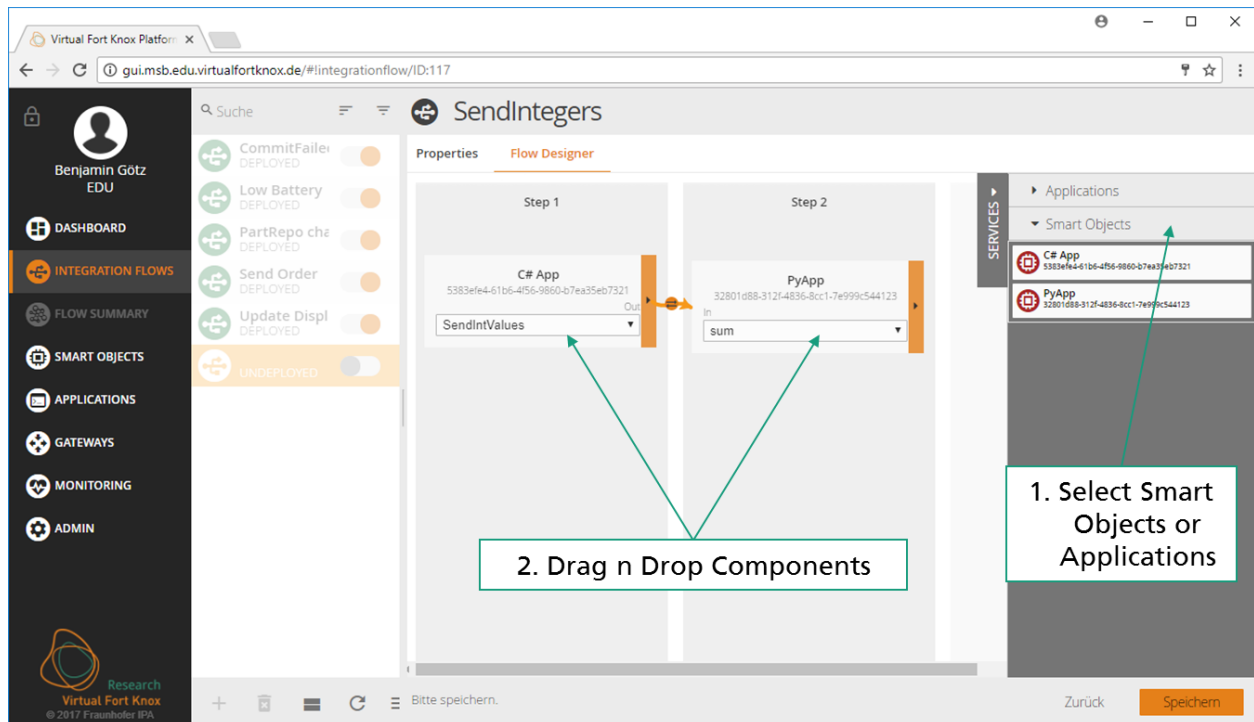
Figure 45: Drag and Drop of Components to initiate the Creation of a new Integration Flow

2.3.3.6.5 Selection of required Events and Functions for each Component

Before connecting the components, the user needs to select the appropriate event and function for components he wishes to connect next. This is done by clicking on the drop down menu for the component and then clicking the desired event or function as indicated in Figure 46. By default, the first event from the list in the self-description is selected for all components. If a component does not supply events, the first function from the list in the self-description is set as default. Selecting a function manually may result in a second drop down menu to appear next to the previous one as can be seen in Figure 47.
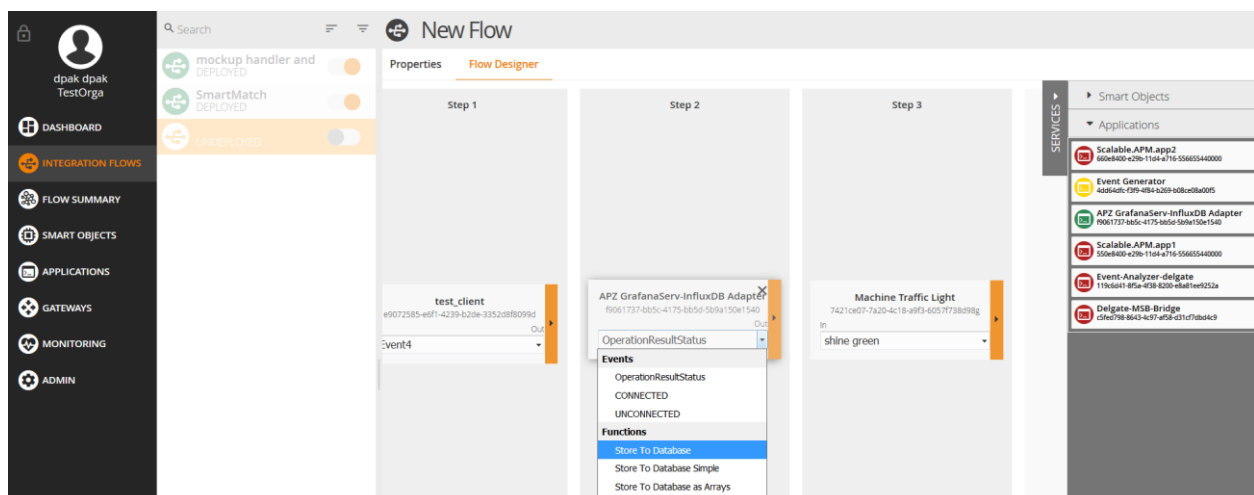
Figure 46: Selection of Event or Function from Drop Down Menu

This implies that the selected function may trigger one of the listed events. The user has to select the desired event from this new list, unless the component is the last one in the chain, where the output event is irrelevant. Due to this behaviour of the GUI it is strongly advised to begin the selection of events and functions at the last component in the desired integration flow.
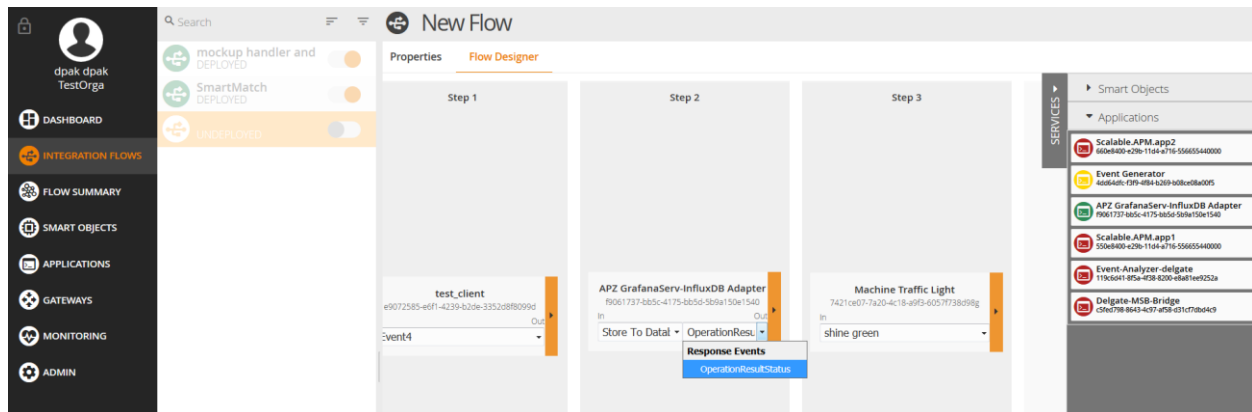
Figure 47: Selection of Return Event based on preselected Function

2.3.3.6.6 Linking of Events to Functions

After the selection process the links between the components have to be set up. Links are always initiated from an event towards a function. The user achieves this by clicking on the orange area of a component with the event and dragging the mouse to the component with the function which he wishes to link to. A successful link is indicated by an orange arrow between the two components, where the arrow is directed towards the component which's function should be executed. An example for successful links can be seen in Figure 48.
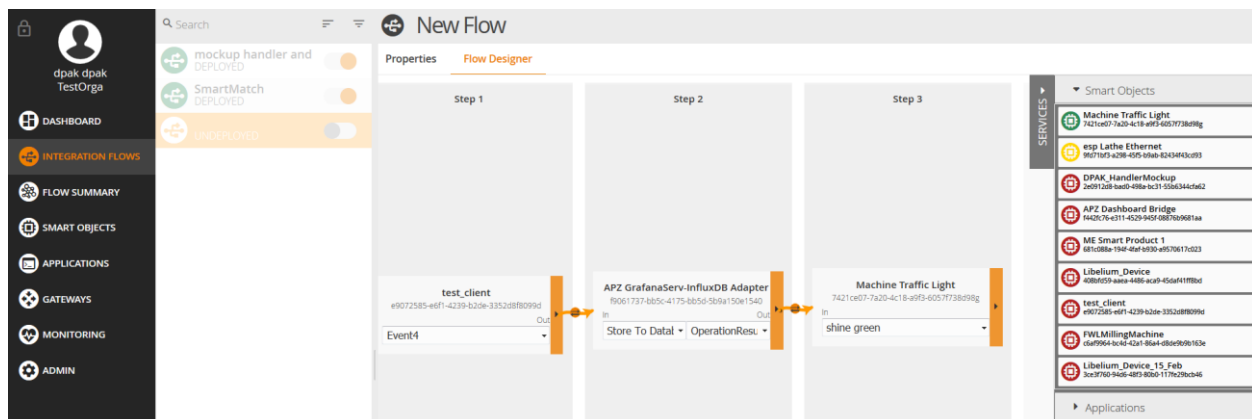
Figure 48: Successfully linked Components

2.3.3.6.7 Mapping of Event Data to Function Parameter Inputs

If the selected functions require no input parameters, specific mapping of parameters can be forgone and the flow is ready to be saved. However, in most cases a mapping of data from the event to the input parameters of the corresponding function is required. The user does this by clicking the small orange dot in the middle of each arrow which will result in a similar GUI to Figure 49. On the left all available data from the event is displayed and on the right all input parameters are shown. For every parameter the particular type is displayed as well.

Figure 49: Detail View for Mapping Event Data to Input Parameters for a Function

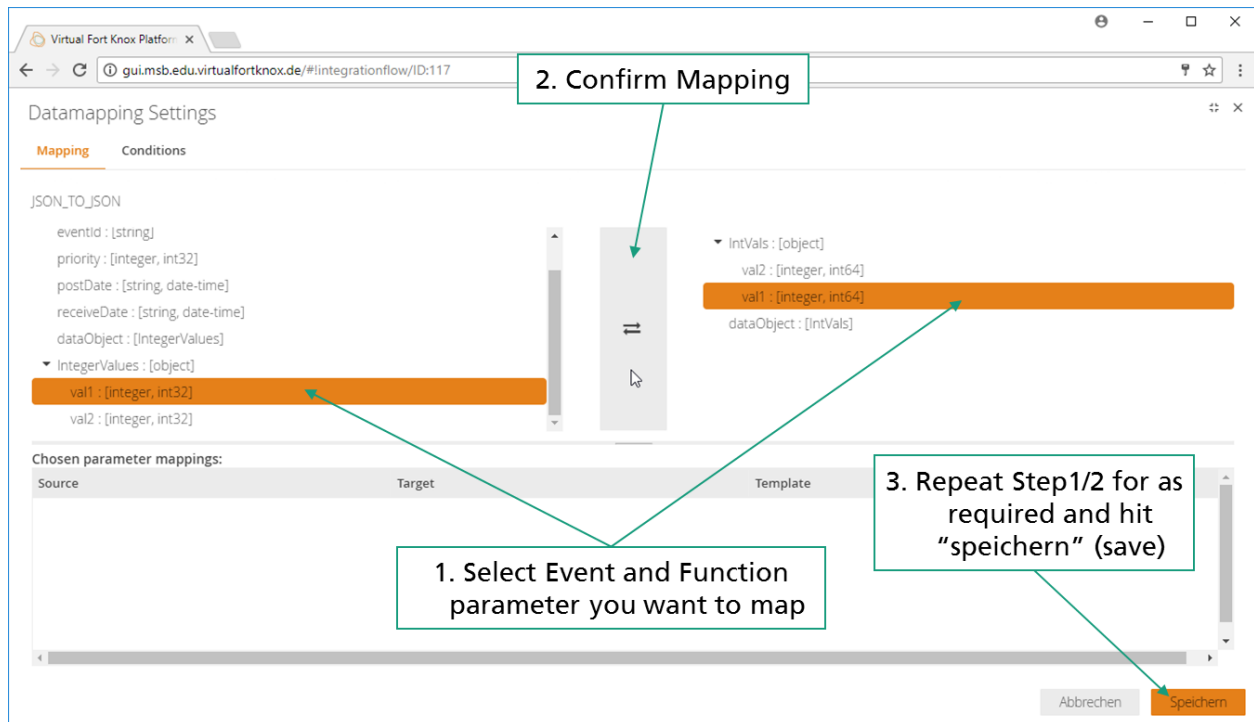The MSB is capable of simple type casts which require no further specification such as integer to float, integer to string. Most input and all output parameters are specific to the component developers. However, the parameters mentioned in Table 9 are always available for mapping on the side of the event. The mapping is conducted by selecting the input and output parameters which are supposed to be mapped, indicated by an orange background and clicking the button with the two arrows in the middle. Once the mapping is completed it will show up in the bottom half of the screen, were it can also be deleted by clicking the icon with the garbage bin. Once the user has mapped all function input parameters to the respective event output parameters, he can click the orange save button in the bottom right to save his changes. In general double mappings should be avoided, as they can lead to confusion. If a double mapping occurs, the latest mapping, indicated by being lower in the list, takes precedence. The mapping process has to be repeated for all links (all orange arrows in Figure 48), where mapping is required.

Table 9: List of standard event properties

uuid	UUID of the component which sends the event.
eventId	ID given to the event by the developer
priority	Priority set by component developer for transfer by the MSB which might be relevant in case of high load. Possible values are: 0-low / 1(default)-medium / 2-high
postDate	Time when event was thrown by the component.
recieveDate	Time when the event was published to the MSB. The distinction is relevant when a component is set up in such a way that it can function autonomously without MSB connection (e.g. in remote regions without WIFI connection. Optionally events can be buffered in this case and published to the MSB once reconnected.

2.3.3.6.8 Setting Conditions for Data Transfer

In some cases, the information of an event should only be forwarded to a function when specific conditions are met, which were not foreseen by the developers who designed the components. In this case the *Conditions* tab in the mapping view can be used to set conditions. Conditions can only be set for the data associated with the current event. Other conditions, for example including information from the previous event, are not configurable. In some cases, smart design of the integration flow using the branching (section 2.3.3.6.9) and merging (section 2.3.3.6.10) can be used to achieve the desired results in combination with conditions. A new condition is set by clicking the relevant input parameter form the associated event and selecting the parameter on the left side and clicking the large button in the middle. This yields a view with a drop down menu like Figure 50 from which the desired comparator can be chosen. Once the user does so, he can set the compare value in a newly appeared field.

Figure 50: View to set Condition which incoming Events are forwarded to the next Component in the Flow.

2.3.3.6.9 Branching of Integration Flows

The length of integration flows, as in the number of event to function links, is largely unlimited. The MSB allows more complex designs as well, beside strictly linear integration flows. It is possible to map one event to several functions of one or more components, effectively creating branches in the flow, as shown in Figure 51. In case a subset of these functions is from the same component, the component needs to be dragged and dropped once for each individual mapping. If the branches do not merge again, the user should decide if two separate integration flows (with two separate names) would improve the overall overview.
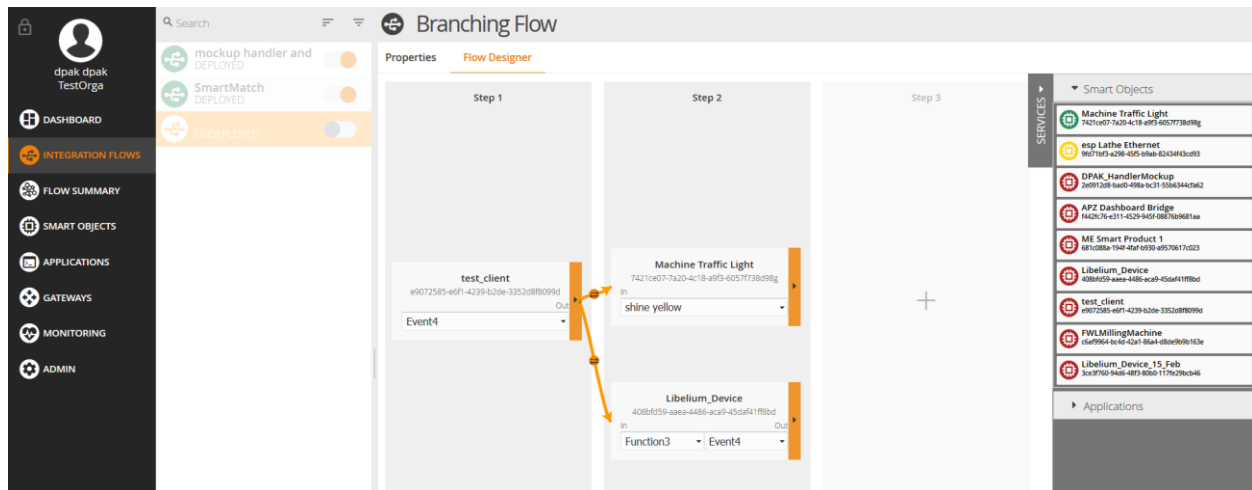
Figure 51: Branching Integration Flow where one Event is forwarded to Two Components

2.3.3.6.10 Merging of Branches in Integration Flows

The merging branches works in a similar fashion as linking and mapping (see Figure 52 for the final view after linking). Once the data of the first event is mapped to the input parameters and saved by clicking the button, the mapping of the data of the second input event can be conducted accordingly. The mappings of the other event will appear in the list of existing mappings (lower half of Figure 49) and vice versa.
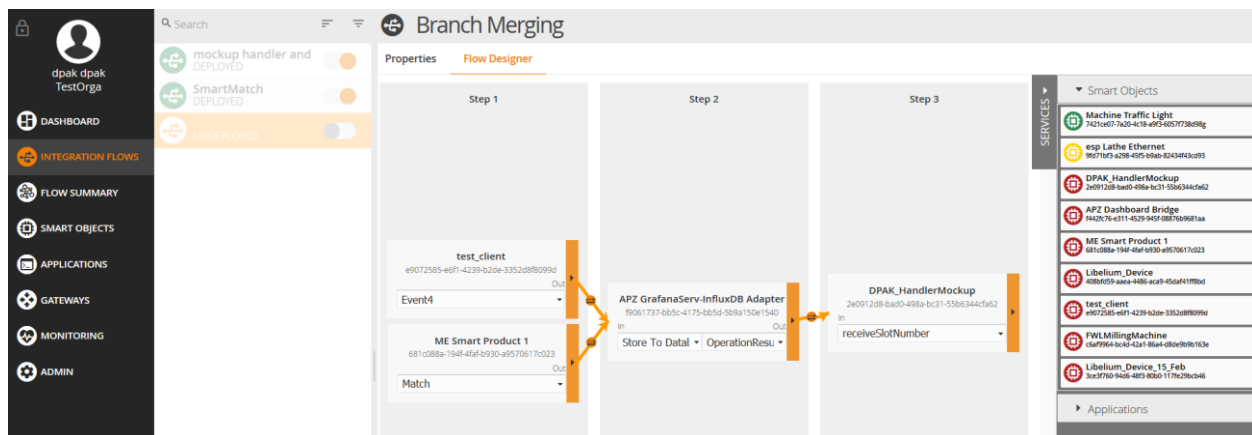

Figure 52: Merging of Branches in an Integration Flow

2.3.3.6.11 Wrap up and Saving the Integration Flow

The final steps before the integration flow can be activated, is saving the current setting and activating the flow. Before doing so, the user should be sure that the steps described before are completed. For better overview they are mentioned here once again:

- Name and description represent the purpose of the integration flow sufficiently.
- All components are in the main area of flow manager (at least one square for each component).
- For each component the correct function / event is selected (lower half of the graphical representation of the component).
- All required links are in place (orange arrows).
- All mappings are set as required for each link.
- All conditions (if required) are set (can only be checked in the respective detail views by clicking the orange nobs in the middle of the arrows and switching to the *Conditions* tab).

If this is the case, the user can click the orange save button on the bottom right of the browser window. As a result, the name of the integration flow in the list of flows on the left half of the screen will change from light gray to dark grey, as seen in Figure 53.
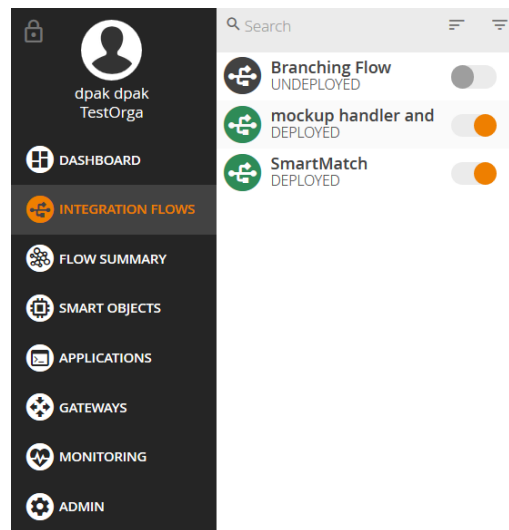
Figure 53: Integration Flow after Saving

By default, the integration flow is still deactivated. To activate it, the user has to hit the toggle button to the right of the name of the integration flow. The status of the flow can also be seen by a quick glance on the color of the icon to the left of the name. Activated flows are indicated by a green icon, while deactivated ones are indicated by a black icon. The activation of the integration flow in the backend of the MSB takes between 1 s – 10 s.

3 Step by step implementation of test scenario for Application Experiment: Health

This section presents a detailed step-by-step guide for integration with the system, which implements the test scenario for Application Experiment: Health, as described in section 1.2.

3.1 A general overview

The system integration can be summarized in just six essential steps; the Figure 54 provides a graphical overview of these steps.
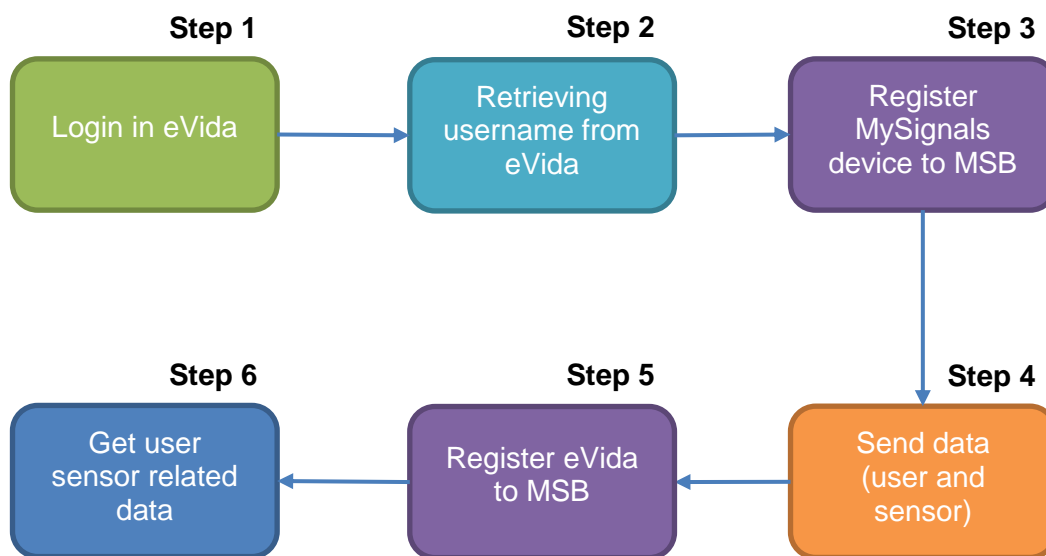
Figure 54: Integration steps

Medical device data is acquired by the means of a sensor integrated with MySignals. MySignals first should login in eVida platform **(Step 1)** so it can retrieve user information **(Step 2)**. After that, it will register the device with Virtual Fort Knox **(Step 3)**, so it can send the sensor data to VFK **(Step 4)**. VFK must them work this data into HL7 format and store it, so eVida can get it **(Step 6)** – after registering to VFK **(Step 5)** – for easy display through the PHR interface.

The technical details of each step are provided in the following points.

3.2 Step 1: Login in eVida

The procedure to login in eVida is described in section 2.1.4.1. The `access_token` obtained should be stored, as it will be needed for Step 2 (3.3).

3.3 Step 2: Retrieving username from eVida

The `username` is obtained as described in 2.1.4.2 (GET logged user in eVida). This value is important, as it will allow matching the sensor data read with the user.

3.4 Step 3: Register MySignals device to MSB

To register the MySignals device as a SmartObject to the MSB a POST request must be sent to https://msb.vfk.de:8083/rest/smartobject/register. The registration process is described in detail in section 2.3.3.1. If the registration has been successful, the HTTP Status Code 201 will be returned and the SmartObject can be verified in the MSB GUI with the token defined in the self-description (as described in section 2.3.2.2). If the POST request returns an ERROR HTTP code or the verification is not possible you should check if the self-description is defined correctly.

POST	https://msb.vfk.de:8083/rest/smartobject/register
Parameters	
Key	*Value*
smartObject	Self description as JSON.

Table 10: POST request to register MySignals device to MSB

3.5 Step 4: Send data

To send data from MySignals device to the MSB an event must be send to the MSB via a POST request to https://msb.vfk.de:8083/rest/data (Table 11). How to send data is described in more detail in section 2.3.3.2.

POST	https://msb.vfk.de:8083/rest/data
Parameters	
Key	*Value*
incomingData	Data as JSON.

Table 11: POST request to send data from MySignals device to MSB

After the data has been received from the MSB, it must be defined where it will be forwarded to. Data routing is defined in the MSB using Integration Flows. In our use case the data of the MySignals device have to be forwarded to a data repository. Therefore, the Integration Flow from Figure 55 must be created. This integration flow is triggered each time an event with sensor data from MySignals device is received. The integration flow forwards the data to the "Data Repository Service" which stores the data in a database. This is just a simple scenario to illustrate the integration into VFK. But it can be easily extended to include any additional services.
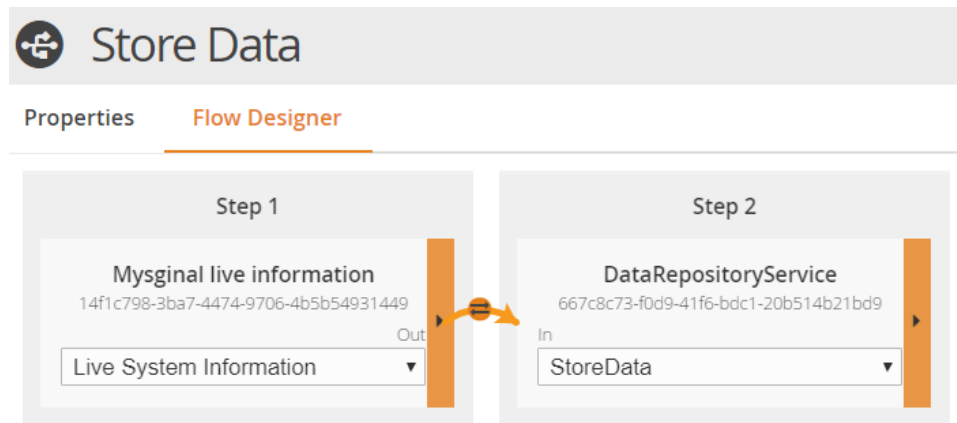
Figure 55: Integration flow to store data in the data repository

3.6 Step 5: Register eVida to MSB

To register the MySignals device as a SmartObject to the MSB a POST request must be sent to https://msb.vfk.de:8083/rest/smartobject/register. The registration process is described in detail in section 2.3.3.1. If the registration has been successful, the HTTP Status Code 201 will be returned and the SmartObject can be verified in the MSB GUI with the token defined in the self-description (as described in section 2.3.2.2). If the POST request returns an ERROR HTTP code or the verification is not possible you should check if the self-description is defined correctly.

POST	https://msb.vfk.de:8083/rest/smartobject/register
Parameters	
Key	*Value*
smartObject	Self description as JSON.

Table 12: POST request to register eVida app to MSB

3.7 Step 6: Get user sensor related data

To request user data from the data repository service running in VFK an event must be send to the MSB via a POST request to https://msb.vfk.de:8083/rest/data (Table 13). How to send data is described in more detail in section 2.3.3.2.

POST	https://msb.vfk.de:8083/rest/data
Parameters	
Key	*Value*
incomingData	Data as JSON.

Table 13: POST request to request user data from data repository service in eVida app

The incoming event triggers the Integration Flow shown in Figure 56 which calls the "RequestData" function of the data repository service. The event data which specifies the requested data, are mapped to the function parameters of that function. After the data repository service executed the request, the data are sent back as response event "ResponseData" to the MSB. This triggers the last step of the

Integration Flow which forwards the response of the request via REST to the eVida app by calling the "ReceiveRequestedData" function of the eVida app. The creation of integration flows is described in detail in section 2.3.3.6.
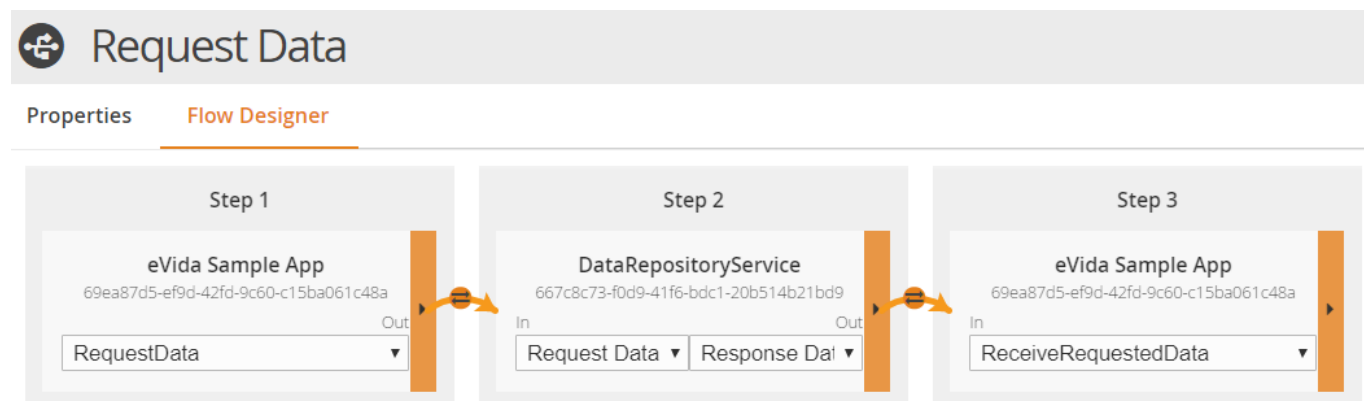

Figure 56: Integration flow to request user data from data repository service in eVida app

4 References

[1]   "Healthy API documentation", 2018. [Online]. Available: https://ge.evida.pt/APIdocumentation; [Accessed: 09-Feb-2018]

[2]   "MySignals, eHealth and Medical IoT Development Platform Technical Guide". Available: http://www.libelium.com/downloads/documentation/mysignals_technical_guide.pdf; [Updated on 12 March 2018]

[3]   "Services Cloud Manager. Cloud Services Guide". [Online]. Available: http://www.libelium.com/downloads/documentation/service_cloud_manager_guide.pdf

[4]   Libelium's IoT Marketplace. [Online]. Available: https://www.the-iot-marketplace.com/cloud-services/mysignals-cloud