Project Number 683612

D 3.3 Third party API call specification

1.0

30 May 2016

Final

Public

petaFuel

Project Partner Contact Information

petaFuel GmbH
Ludwig Adam
Muenchnerstrasse 4
85354 Freising
Germany
Tel: +49 8161 40 60 202
E-Mail: ludwig.adam@petafuel.de

Table of Content

Document Control

Version	Status	Date
0.1	Document outline and first content	9th of May 2016
1.0	Final version	30th of May 2016

Executive Summary

This document consititutes Deliverable 3.3. of work package 3 of the VIMpay project. It specifies third party access on the API and provides use cases and some examples for clarification.

1 Introduction

1.1 Third party usage of the pfREST API

Third party usage denotes all kinds of usage, that is performed outside the VIMpay App on behalf of a certain user ID or card ID.

1.2 Example use cases

There are a couple of scenarios, where such usages are desirable and needed. The following use cases shall illustrate why.

1) The help desk team has to be able to make changes or view into the data upon a request of a user. This is necessary to provide adequate user support and is done via a dedicated platform called SupportFrontend.
2) External developers may develop own extensions for VIMpay. Another use case is integration of VIMpay functionality into other applications.
3) External partners need access to control and look into their customer accounts. For instance, a partner may request or change the address of a certain customer.

1 Concept of third party calls

The authentication levels and the architecture also apply for third party calls and have already been discussed extensively in D 3.1 and D 3.2.

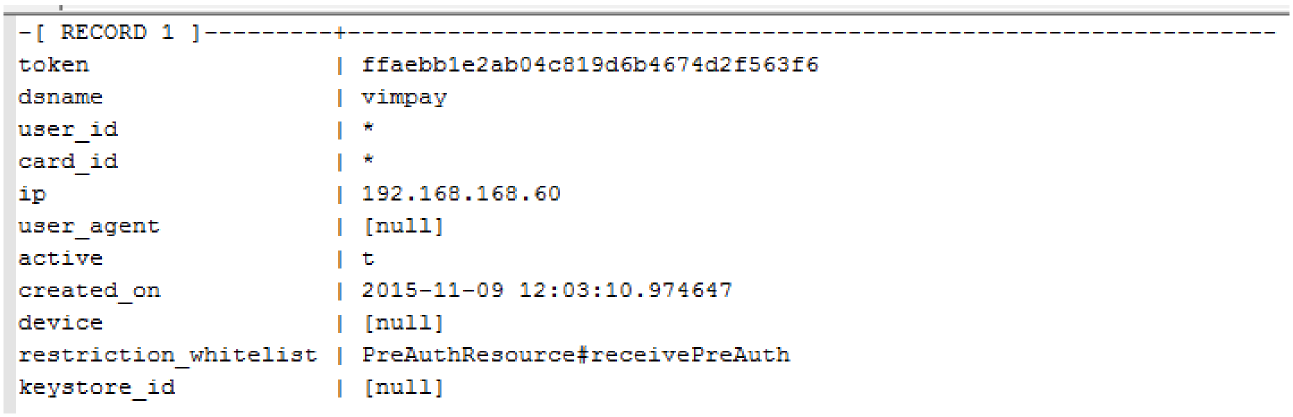
2.1 Restricted access

For security reasons certain API calls require restricted access. Thus in order to be able to use these functions, they need to be put on a whitelist first.

The whitelist is always tied to the token and contains a semicolon separated list of allowed functions. When a restricted function is called, a routine checks the whitelist of the request token and only grants access if a corresponding entry for the called function was found. Please note, that the token also needs to match the Access Control Parameters (DS name, user ID, card ID) by explicitly defining it or by wild card matching.

2.2 Wild card matching

Wild card matching may be used to control multiple accounts with one token. Instead of explicitly defining the Access Control Parameters (DS name, user ID, card ID), an asterisk may be used to match any value. For clarification, consider the following configuration:



```
-[ RECORD 1 ]---------+------------------------------------------------------------
token                 | ffaebb1e2ab04c819d6b4674d2f563f6
dsname                | vimpay
user_id               | *
card_id               | *
ip                    | 192.168.168.60
user_agent            | [null]
active                | t
created_on            | 2015-11-09 12:03:10.974647
device                | [null]
restriction_whitelist | PreAuthResource#receivePreAuth
keystore_id           | [null]
```

By using this token all User IDs and Card IDs of VIMpay may be controlled. Additionally, access is granted to the restricted method PreAuthResource#receivePreAuth due to the entry in the restriction_whitelist. The entry itself represents a method identifier, which consists of a resource and a method. In this case PreAuthResource is the resource and receivePreAuth the method.

2 Setting up third party calls

Before a third party call can be used, a token has to be generated first. After that permissions need to be configured.

3.1 Token generation

The token is represented by a random UUID with having the dashes removed. It is generated by the built-in JDK function UUID.randomUUID(), followed by replacing the dashes. The result is a 32-bit alphanumeric code, e. g. ffaebb1e2ab04c819d6b4674d2f563f6

3.2 Access configuration

Access and permissions are configured and maintained in a database. Before a token can be used, it has to be added to that database. As already mentioned, the token can be tied to a specific product, user ID or card ID or allow more general usage by applying the wildcard character. Functions that reside within AuthLevel.Restricted scope, require an explicit whitelist entry.

3 Examples

For clarification, let’s give some examples:

Request	Configured permissions
GET https://pfrest.pboss.de/rest/user/address	token=ffaebb1e2ab04c819d6b4674d2f563f6
Request parameter	dsname=*
token=ffaebb1e2ab04c819d6b4674d2f563f6	user_id=*
user_id=user123	card_id=*
dsname=vimpay	active=true
	restriction_whitelist=[null]

The above request is valid, since the token is active and the scope is to global due to the wildcard. A whitelist entry is not necessary here, since fetching the user address is not a restricted function.

Let’s have a look to another example, where the request would have failed:

Request	Configured permissions
POST https://pfrest.pboss.de/rest/user/address	token=ffaebb1e2ab04c819d6b4674d2f563f6
Request parameter	dsname=supremacard
token=ffaebb1e2ab04c819d6b4674d2f563f6	user_id=*
user_id=user123	card_id=*
dsname=vimpay	active=true
	restriction_whitelist=[null]

This request is not valid and access will be denied, because the specified token only has permissions to operate on supremacard.

Another valid request using a restricted call might look as follows:

Request	Configured permissions
POST https://pfrest.pboss.de/rest/preauth	token=ffaebb1e2ab04c819d6b4674d2f563f6
Request parameter	dsname=vimpay
token=ffaebb1e2ab04c819d6b4674d2f563f6	user_id=*
user_id=user123	card_id=*
dsname=vimpay	active=true
	restriction_whitelist=PreAuthResource#receivePreAuth

It is important, that the method identifier PreAuthResource#receivePreAuth is included in the token’s permissions. Otherwise the request would have been denied, e. g.

Request	Configured permissions
POST https://pfrest.pboss.de/rest/preauth	token=ffaebb1e2ab04c819d6b4674d2f563f6
Request parameter	dsname=vimpay
token=ffaebb1e2ab04c819d6b4674d2f563f6	user_id=*
user_id=user123	card_id=*
dsname=vimpay	active=true
	restriction_whitelist=[null]