PassBy[ME] Management API Documentation

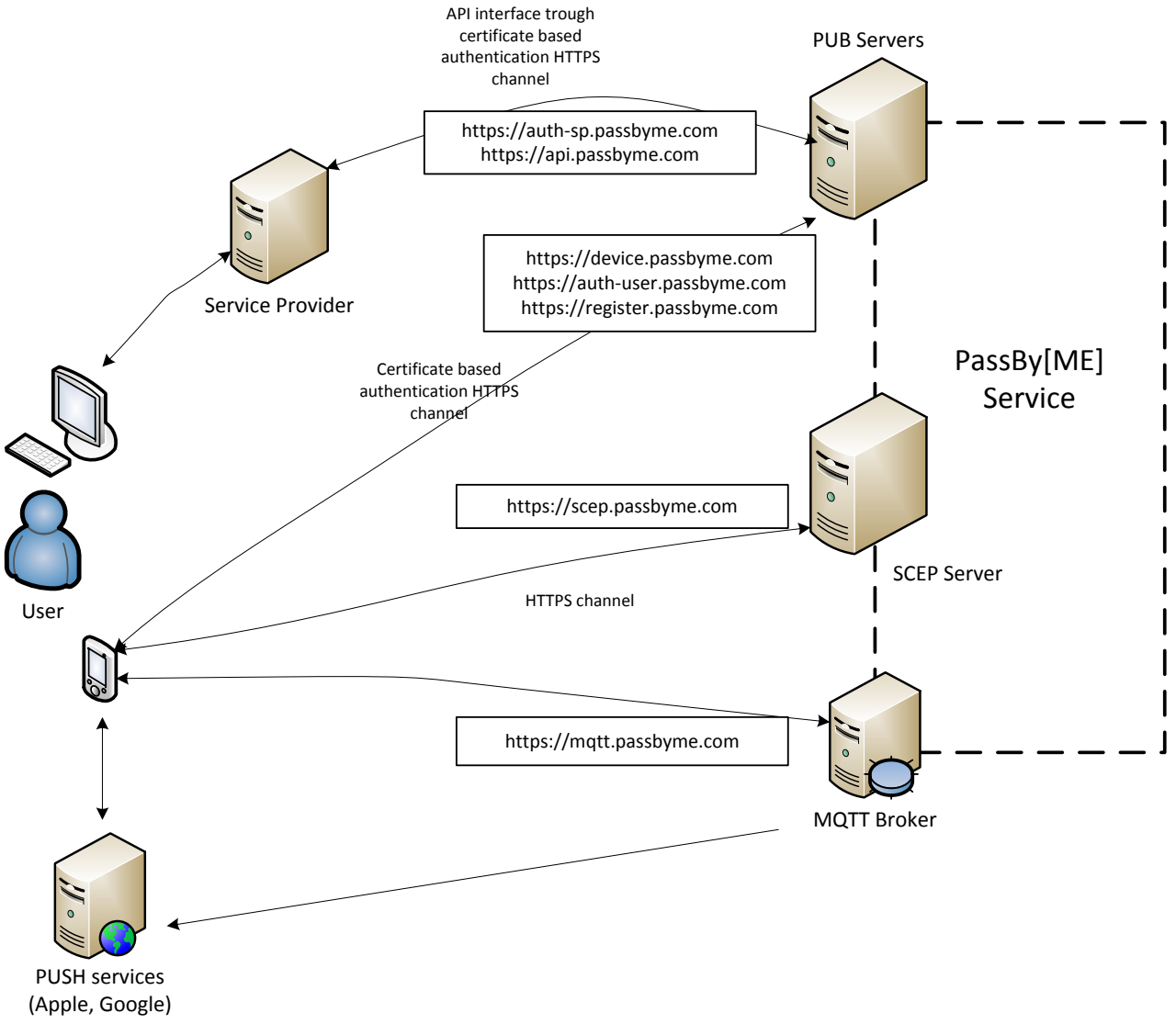
Document id:	PBM_02
Document version:	1.1.12
Author:	Microsec Ltd.
Date:	2016.09.13.
API Version:	**2**

Introduction

This documentation assumes that you are familiar with the basic working model of PassBy[ME] API interface. All information contained herein is of confidential nature and not intended for public distribution.

This document will describe the main technical characteristics to integrate the PassBy[Me] management service.

1.1 PassBy[ME] architecture



Terms

The document uses the following PassBy[ME] specific terms:

Administrator: A human, or machine with access to the Administration interface or the Management API, being able to administer the Organization.

Alias: In the PassBy[ME] system a user can have multiple PassBy[ME] IDs. These IDs are all unique identifiers of the user within the Organization serving as his aliases.

Management API: Programmable webservice interface for administrative operations.

Administration interface: Web based client for administrative operations.

Authentication API: Programmable webservice interface for authentication related operations.

Application: An external application that uses PassBy[ME] as a second factor.

Enrollment: A process that makes the Users mobile device PassBy[ME] ready.

Enrollment sheet: A document that contains vital information to completing the enrollment process.

OID: A globally unique identifier of a User. (Example: 1.2.3.4.5.6.7.8.9)

Organization: The registration unit of the PassBy[ME] system. Encapsulates everything related to a registered parties PassBy[ME] operations, like Administrators, Applications, Users etc.

PassBy[ME] App: The PassBy[ME] mobile application available for iOS and Android in the application stores.

PassBy[ME] ID: A string which is unique to the Organization and identifies a user.

PassBy[ME] ready: A device is PassBy[ME] ready if it has the PassBy[ME] App installed and went through the Enrollment process successfully.

User: Someone who is registered within the Organization willing to use PassBy[ME] as a second factor for authentication while accessing an Application.

UserId: In the context of a User: see Alias. In the context of an Administrator, the UserId is the unique identifier of the Administrator within the Organization. Its value is the email address the Administrator is registered with.

PassBy[ME] Management API

The REST (Representational State Transfer) API provides programmatic access to the management service of PassBy[ME]. The REST API identifies accounts by an authentication certificate-key pair.

You'll need a PassBy[ME] account, and a certificate to access the management API from your application. To get this account data you will need to sign up for a PassBy[ME] test account using the web based administration interface, and register a new organization if you haven't registered before. If you have already registered into the PassBy[ME] service, please log in as an administrator with your previously created credential data (username/password and your PassBy[ME] ready mobile device).

Client certificate authentication is enforced by the server. In order to use the API you are required to use your authentication certificate during all API calls.

3.1 Access to the PassBy[ME] service

Web based administration interface: https://admin.passbyme.com

Authentication API webservice: https://auth-sp.passbyme.com/frontend

Management API webservice: https://api.passbyme.com/register

3.2 Response format

Responses are available in JSON (JavaScript Object Notation) format.

- On success the system returns Strings or Objects represented in JSON format. When accessing PDF documents the binary document is returned.
- PassBy[ME] returns HTTP 420 error code on any error. Error responses are objects represented in JSON format. Error responses contain a *code* that is always present and an optional *message*.

  General error object:

```
{ "code" : <error_code>, "message" : <error_message> }
```

3.2.1 Examples

Successful result with a valid OID as a JSON string:

```
HTTP/1.0 201 OK
"1.3.6.1.4.1.21528.3.3.3.148"
```

Error result:

```
HTTP/1.0 420
{ "code" : "INVALID_SESSION_ID", "message" : "The given session
was not found" }
```

3.3 Authentication

To access the PassBy[ME] service URL you should have a **valid authentication certificate and key (PFX file):** The PassBy[ME] management service URL can be used only with a valid authentication certificate, which you can acquire after the registration process by choosing the followings:

1. Select the 'Organization details' menu
2. Click on the 'Key Details' button
3. Choose 'Download PFX' option

This certificate has a software based private key inside the downloaded PFX. The PFX file is protected with a passphrase, which can be printed on the administration website after a successful registration.

Important: The authentication certificate identifies the registered organization in the PassBy[ME] service.

3.4 Parameters in URL path

Some input parameters of the PassBy[ME] REST interface are included in the request path. These parameters are distinguished from other path values by a preceding colon. For example:

```
PUT /rest/method/with/a/:parameter/and/an/:otherparameter
```

contains two parameters: "parameter" and "otherparameter".

3.5 API Version

The PassBy[ME] service requires the client to declare its supported API version. To do that, the client has to include the **X-PBM-API-VERSION** custom HTTP header into **every request** with the value of its current API version. The current API version this document is compatible with can be found on the cover of this document.

Example:

```
X-PBM-API-VERSION: 1
```

User Management API

4.1 Creating a user

Method:
POST /rest/users

Creates a new PassBy[ME] user and returns the OID of the new user.

Note: this function does not automatically create a new enrollment for the newly created user. To create new enrollments for users see: 4.7 Create new enrollment.

4.1.1 Input parameters

The input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
userId	The PassBy[ME] ID of a user.	required
email	Email address of the user.	required
fullName	Full name of the user.	optional
phoneNumber	The phone number of the user.	optional

4.1.2 Output

Outputs the created user or one of the following errors:

- ALREADY_EXISTS - if the given userId already exists.
- INSUFFICIENT_BALANCE – if you don't have enough credit to create the user

4.1.3 Example request

```
POST /rest/users
{ "userId": "user01", "email": "user01@email.com", "fullName":
"User 01", "phoneNumber": "12345678" }
```

4.1.4 Example Response

```
HTTP/1.0 200 OK
```

```
{ "oid": "1.3.6.1.4.1.21528.3.3.3.1588", "fullName": "fullName",
"email": "user01@email.com", "phoneNumber": "12345678" }
```

or

```
HTTP/1.0 420
{ "code": "ALREADY_EXISTS" }
```

4.2 Get the list of users

Method:
GET /rest/users

Returns the list of PassBy[ME] users.

4.2.1 Input parameters

None.

4.2.2 Output

A JSON encoded array of *UserData* objects that contains the list of users under the "data" key.

4.2.3 Example request

```
GET /rest/users
```

4.2.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"oid":"1.3.6.1.4.1.21528.3.3.3.109","shortOID":"3.3.109
","fullName":"User
01","email":"user01@email.hu","phoneNumber":"123456",
"disabled":false},
{"oid":"1.3.6.1.4.1.21528.3.3.3.112","shortOID":"3.3.112","fullNa
me":"User 02","email":"user02@email.hu","phoneNumber":"",
"disabled":false}]}
```

4.3 Get the number of users

Method:
GET /rest/users/count

Get the number of users in the account.

4.3.1 Input parameters:

None.

4.3.2 Output

Outputs the number of users.

4.3.3 Example request

```
GET /rest/users/count
```

4.3.4 Example response

```
HTTP/1.0 200 OK
12
```

4.4 Get the data of a given user

Method:
GET /rest/users/:oid

Find the user with the given OID.

4.4.1 Input parameters

- oid - The OID of the user.

4.4.2 Output

Outputs a JSON encoded *UserData* object, containing the data of the user with the given OID, or one of the following errors:

- NOT_FOUND - if the given oid was not found.

4.4.3 Example request

```
GET /rest/users/1.3.6.1.4.1.21528.3.3.3.109
```

4.4.4 Example Response

```
HTTP/1.0 200 OK
{"oid":"1.3.6.1.4.1.21528.3.3.3.109","shortOID":"3.3.109","fullNa
me":"User 01","email":"user01@email.hu","phoneNumber":"123456",
"disabled": false}
```

or

```
HTTP/1.0 420
{ "code": "NOT_FOUND" }
```

4.5 Delete a user

Method:
DELETE /rest/users/:oid

Deletes the user with the given OID.

4.5.1 Input parameters

- oid - The OID of the user to be deleted.

4.5.2 Output

Answers HTTP OK, or one of the following errors:

- NOT_FOUND - if the user with the given OID was not found.

4.5.3 Example request

```
DELETE /rest/users/1.3.6.1.4.1.21528.3.3.3.109
```

4.5.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{ "code": "NOT_FOUND" }
```

4.6 Modify user

Method:
PUT /rest/users/:oid

Modify the user with the given OID.

4.6.1 Input parameters

- oid - The OID of the selected user.

The rest of the input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
email	The new email address of the user.	required
fullName	The new full name of the user.	optional
phoneNumber	The new phone number of the user.	optional
disabled	True if the user is disabled, false otherwise.	optional

4.6.2 Output

Outputs HTTP 200 on success, or one of the following errors:

- NOT_FOUND - if the given OID was not found.

4.6.3 Example request

```
PUT /rest/users/1.3.6.1.4.1.21528.3.3.3.109
{ "email": "user02@email.hu", "fullName": "User 02",
"phoneNumber": "12345678" }
```

4.6.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"1.3.6.1.4.1.21528.3.3.3.109 in
Account: 1.3.6.1.4.1.21528.3.3.2.100 (Test Organization)"}
```

4.7 Create new enrollment

Method:
POST /rest/users/:oid/enrollments

Create a new enrollment for the given user.

4.7.1 Input parameters

- oid - The OID of the user.

4.7.2 Output

Outputs the created enrollment, or one of the following errors:

- NOT_FOUND - if the given OID was not found.

4.7.3 Example requests

```
POST /rest/users/1.3.6.1.4.1.21528.3.3.3.102/enrollments
```

4.7.4 Example Response

```
HTTP/1.0 200 OK
{"enrollmentId": "9V45pdVWTIREkXWa", "expireDate": "2015-02-
26T15:54:22.696Z", "state": "ACTIVE", "sent": "false",
"deviceConfigUrl": "https://passbyme.com/
deviceConfig/1.3.6.1.4.1.21528.3.3.2.156/URPvk4EuqVJSUfB3?expireD
ate\u003d2015-04-23T10%3A24%3A22.893Z"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"1.3.6.1.4.1.21528.3.3.3.102"}
```

4.8 List enrollments of user

Method:
GET /rest/users/:oid/enrollments

Returns the active enrollments of the given user.

4.8.1 Authentication type

Client certificate authentication

4.8.2 Input parameters

- oid - The OID of the user.

4.8.3 HTTP response code

Code	
200 - Ok	Success.
420 - Application error	See the error codes below.

4.8.4 Output

A JSON encoded array of EnrollmentData objects that contains the list of enrollments under the "data" key.

- NOT_FOUND - if the given userId was not found.

4.8.5 Example request

```
GET /rest/users/1.3.6.1.4.1.21528.3.3.3.102/enrollments
```

4.8.6 Example Response

```
HTTP/1.0 200 OK
{"data":[{"enrollmentId": Ot5pL2mkRVO6LYav, "expireDate": "2014-
12-10T16:00:15.065", "state": "ACTIVE", "deviceConfigUrl":
"https://passbyme.com/
deviceConfig/1.3.6.1.4.1.21528.3.3.2.156/URPvk4EuqVJSUfB3?expireD
ate\u003d2015-04-23T10%3A24%3A22.893Z"},
{"enrollmentId": "aT7xL2mkRuO6LYiK", "expireDate": "2014-12-
10T16:00:09.560", "state": "ACTIVE",
"deviceConfigUrl": "https://passbyme.com/
deviceConfig/1.3.6.1.4.1.21528.3.3.2.156/URPvk4EuqVJSUfB3?expireD
ate\u003d2015-04-23T10%3A24%3A22.893Z"
}]}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"1.3.6.1.4.1.21528.3.3.3.102"}
```

4.9 Download enrollment

Method:
GET /download/users/:oid/enrollments/:enrollmentId/pdf

Downloads the enrollment in pdf format.

4.9.1 Input parameters

- oid - The OID of the user.
- enrollmentId - The ID of the enrollment.

4.9.2 Output

The downloadable enrollment sheet in pdf format, or one of the following error codes:

- NOT_FOUND - if the given enrollment was not found.

4.9.3 Example request

```
GET
/download/users/1.3.6.1.4.1.21528.3.3.3.102/enrollments/Ot5pL2mkR
VO6LYav/pdf
```

4.9.4 Example Response

```
HTTP/1.0 200 OK
<PDF bytes>
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"TqMmAt4VDNRws60E"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"1.3.6.1.4.1.21528.3.3.3.102"}
```

4.10 Send enrollment in email

Method:
POST /rest/users/:oid/enrollments/:enrollmentId/pdfviaemail

Send the specified enrollment pdf document to the owner user via e-mail. The user's e-mail address is used as recipient.

4.10.1 Input parameters

- oid - The OID of the owner user.
- enrollmentId – The id of the enrollment.

4.10.2 Output

Outputs HTTP 200 on success, or one of the following errors:

- NOT_FOUND - if the given oid or enrollmentId was not found.

4.10.3 Example requests

```
POST
/rest/users/1.3.6.1.4.1.21528.3.3.3.102/enrollments/TqMmAt4VDNRws
60E/pdfviaemail
```

4.10.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"TqMmAt4VDNRws60E"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"1.3.6.1.4.1.21528.3.3.3.102"}
```

4.11 Delete enrollment

Method:
DELETE /rest/users/:oid/enrollments/:enrollmentId

Deletes the enrollment specified by the enrollmentId of the user specified by the OID.

4.11.1 Input parameters

- oid - The OID of the owner of the enrollment.
- enrollmentId - The enrollmentId of the enrollment to be deleted.

4.11.2 Output

Outputs HTTP 200 on success or one of the following errors:

- NOT_FOUND - if the given OID or enrollment was not found.

4.11.3 Example request

```
DELETE
/rest/users/1.3.6.1.4.1.21528.3.3.3.109/enrollments/sx8Ufy1H7AMXo
qrC
```

4.11.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"Enrollment with id
sx8Ufy1H7AMXoqrC not exists!"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"1.3.6.1.4.1.21528.3.3.3.109"}
```

4.12 Create LoginName for user

Method:
POST /rest/loginNames/:oid

Adds a new userId (alias) for the specified PassBy[ME] user. Returns the new userId.

4.12.1 Input parameters

- oid - The OID of the chosen user, who receives the alias

The the rest of the input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
userId	The PassBy[ME] ID of a user.	required

4.12.2 Output

Outputs the created alias on success, or one of the following errors:

- ALREADY_EXISTS - if the given userId already exists.

4.12.3 Example request

```
POST /rest/loginNames/1.3.6.1.4.1.21528.3.3.3.112
{ "userId": "alias01" }
```

4.12.4 Example Response

```
HTTP/1.0 200 OK
{"userId":"alias01","oid":"1.3.6.1.4.1.21528.3.3.3.112"}
```

or

```
HTTP/1.0 420
{"code":"ALREADY_EXISTS","message":"User login already exists:
LoginName{userId="alias01", orgId="1.3.6.1.4.1.21528.3.3.2.100",
oid=1.3.6.1.4.1.21528.3.3.3.112}"}
```

4.13 Get LoginNames of user

Method:
GET /rest/loginNames/:oid

Returns the list of userIds (aliases) of the specified PassBy[ME] user.

4.13.1 Input parameters

- oid - OID of the user.

4.13.2 Output

A JSON encoded array of LoginNameData objects. Each LoginNameData object represents the same user with a given alias.

4.13.3 Example request

```
GET /rest/loginNames/1.3.6.1.4.1.21528.3.3.3.112
```

4.13.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"userId":"userId01","oid":"1.3.6.1.4.1.21528.3.3.3.112"
,"shortOID":"3.3.112"},
{"userId":"alias01","oid":"1.3.6.1.4.1.21528.3.3.3.112","shortOID
":"3.3.112"}]}
```

4.14 Find LoginName by userId

Method:
GET /rest/loginNames

Find the loginName with the given userId.

4.14.1 Input parameters

- userId query parameter – One of the aliases of the user.

4.14.2 Output

Outputs a JSON encoded *LoginName* object, or one of the following errors:

- NOT_FOUND - if the given alias was not found.

4.14.3 Example request

```
GET /rest/loginNames?userId=user01
```

4.14.4 Example Response

```
HTTP/1.0 200 OK
{"oid":"1.3.6.1.4.1.21528.3.3.3.109","shortOID":"3.3.109","userId
":"user01"
```

or

```
HTTP/1.0 420
{ "code": "NOT_FOUND" }
```

4.15 Delete LoginName

Method:
DELETE /rest/loginNames

Deletes the specified userId (alias) of the specified PassBy[ME] user.

4.15.1 Input parameters

- oid query parameter - The oid of the PassBy[ME] user who owns the userId to be deleted.
- userId query parameter parameter - The userId to be deleted.

4.15.2 Output

Outputs HTTP 200 on success or one of the following error:

- NOT_FOUND - if the user does not own the specified userId.

4.15.3 Example request

```
DELETE
/rest/loginNames?oid=1.3.6.1.4.1.21528.3.3.3.112&userId=alias01
```

4.15.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"No rows were deleted while
deleting 1.3.6.1.4.1.21528.3.3.3.112/alias01!"}
```

Administrator Management API

5.1 List administrators

Method:
GET /rest/administrators

Returns the list of administrators.

5.1.1 Input parameters

None.

5.1.2 Output

A JSON encoded array of AdministratorData objects that contains the list of administrators under the "data" key.

5.1.3 Example request

```
GET /rest/administrators
```

5.1.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"userId":"admin@microsec.hu","fullName":"Admin
name","phoneNumber":"123456",”oid”:”1.2.3.4.5.6.7”}]}
```

5.2 Add administrator as user

POST /rest/administrators/user

Add an existing administrator as a new PassBy[ME] user.

5.2.1 Input parameters

The input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
userId	The user will be created with this PassBy[ME] ID.	required
adminId	Same as the login name (email address)	required

5.2.2 Output

Outputs the created user, or one of the following errors:

- ALREADY_EXISTS - if the given administrator has already been added to the PassBy[ME] users.

5.2.3 Example request

```
POST /rest/users
{ userId: "login01", adminId: "admin@email.com" }
```

5.2.4 Example Response

```
HTTP/1.0 200 OK
{"oid": "1.3.6.1.4.1.21528.3.3.3.157", "fullName": "Admin user",
"email": "admin@email.com", "phoneNumber": "123456"}
```

or

```
HTTP/1.0 420
{"code":"ALREADY_EXISTS","message":"Customer with id:
1.3.6.1.4.1.21528.3.3.3.114 already exists"}
```

5.3 List active invitations

Method:
GET /rest/invitations

Returns the list of active invitations.

5.3.1 Input parameters

None.

5.3.2 Output

A JSON encoded array of InvitationData objects that contains the list of invitations under the "data" key.

5.3.3 Example request

```
GET /rest/administrators
```

5.3.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"expiryDate":"2015-01-
30T03:09:47.073","code":"BUmuaZBat8BkNhtk"}]}
```

5.4 Create a new invitation

Method:
POST /rest/invitations

Create a new invitation.

5.4.1 Input parameters

None.

5.4.2 Output

Outputs the created invitation, or one of the following errors:

- PRICING_LIMIT_INVITATION - if your subscription does not allow more invitations.

5.4.3 Example request

```
POST /rest/invitations
```

5.4.4 Example Response

```
HTTP/1.0 200 OK
{"expiryDate": "2015-02-26T03:51:38.340Z", "code":
"sSnj6R9NP8SyANbm"}
```

or

```
HTTP/1.0 420
{"code":"PRICING_LIMIT_INVITATION","message":"You have exceeded
the maximum number of invitations"}
```

5.5 Delete an invitation

Method:
DELETE /rest/invitations/:invitationcode

Delete an invitation code.

5.5.1 Input parameters

- Invitation code to be deleted.

5.5.2 Output

Outputs HTTP 200.

5.5.3 Example request

```
DELETE /rest/invitations/sSnj6R9NP8SyANbm
```

5.5.4 Example Response

```
HTTP/1.0 200 OK
```

5.6 Create new enrollment for an administrator

Method:
POST /rest/administrators/:userId/enrollments

Create a new enrollment for the given administrator.

5.6.1 Input parameters

- userId - The userId of the administrator.

5.6.2 Output

Outputs the created enrollment, or one of the following errors:

- NOT_FOUND - if the given userId was not found.

5.6.3 Example requests

```
POST /rest/administrators/admin01@gmail.com/enrollments
```

5.6.4 Example Response

```
HTTP/1.0 200 OK
{"enrollmentId":"G1ehaAwieWjikPsv","expireDate":"2015-02-
26T15:53:00.263Z","state":"ACTIVE","sent":"false"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"admin01@gmail.com"}
```

5.7 Get enrollments of administrator

Method:
GET /rest/administrators/:userId/enrollments

Returns the active enrollments of the given administrator.

5.7.1 Input parameters

- userId - The userId of the administrator.

5.7.2 Output

A JSON encoded array of EnrollmentData objects that contains the list of enrollments under the "data" key.

- NOT_FOUND - if the given userId was not found.

5.7.3 Example request

```
GET /rest/administrators/admin01@gmail.com/enrollments
```

5.7.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"enrollmentId":Ot5pL2mkRVO6LYav,"expireDate":"2014-12-
10T16:00:15.065","state":"ACTIVE"},{"enrollmentId":aT7xL2mkRuO6LY
iK,"expireDate":"2014-12-10T16:00:09.560","state":"ACTIVE"}]}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"admin01@gmail.com"}
```

5.8 Download administrator enrollment PDF

Method:
GET
/download/administrators/:userId/enrollments/:enrollmentId/pdf

Downloads the enrollment in pdf format.

5.8.1 Input parameters

- userId - The userId of the administrator.
- enrollmentId - The id of the enrollment.

5.8.2 Output

The downloadable enrollment sheet in pdf format or one of the following error codes:

- NOT_FOUND - if the given enrollment was not found.

5.8.3 Example request

```
GET
/rest/administrators/admin01@gmail.com/enrollments/Ot5pL2mkRVO6LY
av/pdf
```

5.8.4 Example Response

```
HTTP/1.0 200 OK
<PDF bytes>
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"TqMmAt4VDNRws60E"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"admin01@gmail.com"}
```

5.9 Send enrollment to admin in email

Method:
POST
/rest/administrators/:userId/enrollments/:enrollmentId/pdfviaemail

Send the specified enrollment pdf document to the owner administrator via e-mail. The administrator's userId is used as recipient.

5.9.1 Input parameters

- userId - The userId of the owner administrator.
- enrollmentId - Id of the enrollment.

5.9.2 Output

Outputs HTTP 200, or one of the following errors:

- NOT_FOUND - if the given userId or enrollmentId was not found.

5.9.3 Example requests

```
POST
/rest/administrators/admin01@gmail.com/enrollments/TqMmAt4VDNRws6
0E/pdfviaemail
```

5.9.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"TqMmAt4VDNRws60E"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"admin01@gmail.com"}
```

5.10 Delete administrator enrollment

Method:
DELETE /rest/administrators/:userId/enrollments/:enrollmentId

Deletes the enrollment specified by the enrollmentId of the administrator specified by the userId.

5.10.1 Input parameters

- userId - The userId of the owner of the enrollment.
- enrollmentId: The enrollmentId of the enrollment to be deleted.

5.10.2 Output

Outputs HTTP 200 or one of the following errors:

- NOT_FOUND - if the given userId or enrollment was not found.

5.10.3 Example request

```
DELETE
/rest/administrators/admin01@gmail.com/enrollments/sx8Ufy1H7AMXoq
rC
```

5.10.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"Enrollment with id
sx8Ufy1H7AMXoqrC not exists!"}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"admin01@gmail.com"}
```

Application Management API

6.1 Create application

Method:
POST /rest/applications

Creates a new Application registration. Returns the identifier of the new application.

6.1.1 Input parameters

The input must be POSTed in JSON format as a request body. The JSON object consists of the following key.

JSON key		
name	Name of the application.	required

6.1.2 Output

Outputs the new application.

6.1.3 Example request

```
POST /rest/applications
{ "name": "application01" }
```

6.1.4 Example Response

```
HTTP/1.0 200 OK
{"id": "56838c64-67ef-48f1-a414-70255cd0439c", "name":
"application01"}
```

6.2 List applications

Method:
GET /rest/applications

Returns the list of applications registered in the PassBy[ME] system.

6.2.1 Input parameters

None.

6.2.2 Output

A JSON encoded array of ApplicationData objects that contains the list of applications under the "data" key.

6.2.3 Example request

```
GET /rest/applications
```

6.2.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"id":"aacf6f4d-2f31-44e8-b212-
d1328070e11d","name":"application01","organizationId":"1.3.6.1.4.
1.21528.3.3.2.100"},
{"id":"5b23d69e-180b-40e3-a3e2-
8163dcea34b9","name":"application02","organizationId":"1.3.6.1.4.
1.21528.3.3.2.100"}]}
```

6.3 Get application data

Method:
GET /rest/applications/:applicationId

Find the application with the given applicationId.

6.3.1 Input parameters

- applicationId - The id of the application in the PassBy[ME] system.

6.3.2 Output

Outputs a JSON encoded ApplicationData object containing all the data of the application with the given applicationId, or one of the following errors:

- NOT_FOUND - if the given applicationId was not found.

6.3.3 Example request

```
GET /rest/applications/aacf6f4d-2f31-44e8-b212-d1328070e11d
```

6.3.4 Example Response

```
HTTP/1.0 200 OK
{"data":{"id":"aacf6f4d-2f31-44e8-b212-
d1328070e11d","name":"application01","organizationId":"1.3.6.1.4.
1.21528.3.3.2.100"}}
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"Application with id aacf6f4d-2f31-
44e8-b212-d1328070e11d not exists organization
1.3.6.1.4.1.21528.3.3.2.100 !"}
```

6.4 Delete application

Method:
DELETE /rest/applications/:applicationId

Deletes the application from the PassBy[ME] system with the given applicationId.

6.4.1 Input parameters

- applicationId - The applicationId of the application to be deleted.

6.4.2 Output

Outputs HTTP 200 on success or one of the following errors:

- NOT_FOUND - if the application with the given applicationId was not found.

6.4.3 Example request

```
DELETE /rest/applications/aacf6f4d-2f31-44e8-b212-d1328070e11d
```

6.4.4 Example Response

```
HTTP/1.0 200 OK
"aacf6f4d-2f31-44e8-b212-d1328070e11d"
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"Application with id aacf6f4d-2f31-
44e8-b212-d1328070e11d not exists organization
1.3.6.1.4.1.21528.3.3.2.100 !"}
```

6.5 Modify application

Method:
PUT /rest/applications/:applicationId

Modify the application with the given applicationId.

6.5.1 Input parameters

- applicationId - The applicationId of the application to be modified.

The rest of the input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
name	The new name of the application.	required

6.5.2 Output

Outputs the identifier of the successfully modified application, or one of the following errors:

- NOT_FOUND - code if the given applicationId was not found.

6.5.3 Example request

```
PUT /rest/applications/aacf6f4d-2f31-44e8-b212-d1328070e11d
{ "name": "Modified Application" }
```

6.5.4 Example Response

```
HTTP/1.0 200 OK
```

or

```
HTTP/1.0 420
{"code":"NOT_FOUND","message":"Application with id aacf6f4d-2f31-
44e8-b212-d1328070e11d not exists organization
1.3.6.1.4.1.21528.3.3.2.100 !"}
```

Device Management API

7.1 List all devices of users

Method:
GET /rest/devices

Returns all devices.

7.1.1 Input parameters

None.

7.1.2 Output

A JSON encoded array of DeviceData objects that contains the list of users under the "data" key.

7.1.3 Example request

```
GET /rest/devices
```

7.1.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"deviceName":"User 01
phone","vendorId":"vendorId1","customerFullName":"User
01","deviceType":"ANDROID"},
{"deviceName":"User 02
phone","vendorId":"vendorId2","customerFullName":"User
02","deviceType":"APPLE"}]}
```

7.2 List devices of user

Method:
GET /rest/users/:oid/devices

Returns the devices of the given user.

7.2.1 Input parameters

- oid - The OID of the user.

7.2.2 Output

A JSON encoded array of DeviceData objects that contains the list of devices under the "data" key, or one of the following errors:

- NOT_FOUND - if the given OID was not found.

7.2.3 Example request

```
GET /rest/users/1.3.6.1.4.1.21528.3.3.3.102/devices
```

7.2.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"deviceName":"User 01
phone","vendorId":"vendorId1","customerFullName":"User
01","deviceType":"ANDROID"},
{"deviceName":"User 02
phone","vendorId":"vendorId2","customerFullName":"User
02","deviceType":"APPLE"}]}
```

7.3 List devices of administrator

Method:
GET /rest/administrators/:userId/devices

Returns the devices of the given administrator.

7.3.1 Input parameters

- userId - The userId of the administrator.

7.3.2 Output

A JSON encoded list of DeviceData objects that contains the list of devices under the "data" key, or one of the following errors:

- NOT_FOUND - if the given userId was not found.

7.3.3 Example request

```
GET /rest/administrators/admin01@gmail.com/devices
```

7.3.4 Example Response

```
HTTP/1.0 200 OK
{"data":[{"deviceName":"Admin 01
phone","vendorId":"vendorId1","customerFullName":"Admin
01","deviceType":"ANDROID"},
{"deviceName":"Admin 02
phone","vendorId":"vendorId2","customerFullName":"Admin
02","deviceType":"APPLE"}]}
```

7.4 Delete device

Method:
POST /rest/devices/deactivations

Deletes a device.

7.4.1 Input parameters

The rest of the input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
suspensionPassword	The suspension password of the owner of the device.	required

7.4.2 Output

Outputs HTTP 200 on success, or one of the following errors:

- NOT_FOUND - if the given device was not found.
- FORBIDDEN - if the suspension password is invalid.

7.4.3 Example request

```
DELETE /rest/devices/vendorIdOfDevice/12345678
```

7.4.4 Example Response

```
HTTP/1.0 200 OK
```

7.5 Resend suspension password

Method:
POST /rest/devices/suspensionpass

Resends suspension password via email.

7.5.1 Input parameters

The input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
vendorId	The vendorId of the device which belongs to the suspension password to resend.	required

7.5.2 Output

Outputs HTTP 200, or one of the following errors:

- NOT_FOUND - if the given device was not found.

7.5.3 Example request

```
POST /rest/devices/suspensionpass
{"vendorId": "vendorIdOfDevice"}
```

7.5.4 Example Response

```
HTTP/1.0 200 OK
```

Organization Management API

8.1 Get organization details

Method:
GET /rest/organization

Returns organization details.

8.1.1 Input parameters

None

8.1.2 Output

A JSON encoded OrganizationData object.

8.1.3 Example request

```
GET /rest/organization
```

8.1.4 Example Response

```
HTTP/1.0 200 OK
{"organizationId": "1.2.3.4", name: "Test Org.", "pricing" :
"Free", "email", "support@testorg.com",
"enrollmentExpirationHours" : 24, "invitationExpirationHours" :
12, "confirmationLetterEnabled": true}
```

8.2 Update organization

Method:
PUT /rest/organization

Updates organization details.

8.2.1 Input parameters

The input must be POSTed in JSON format as a request body. The JSON object consists of the following keys.

JSON key		
email	Email address of the organization.	required
name	The name of the Organization.	required
enrollmentExpirationHours	The expiration of the generated enrollment sheets in hours.	required
invitationExpirationHours	The expiration of the generated invitations in hours.	required
confirmationLetterEnabled	Determines whether confirmation letter sending is enabled to PassBy[ME] users.	required

8.2.2 Output

Outputs HTTP 200, or one of the following errors:

- NOT_FOUND - if the given organization was not found.

8.2.3 Example request

```
PUT /rest/organization
{"email" : "newmail@testorg.hu", "name": "Test Organization",
"enrollmentExpirationHours": 48, "invitationExpirationHours": 24,
"confirmationLetterEnabled": true}
```

8.2.4 Example Response

```
HTTP/1.0 200 OK
```

8.3 Get account limitations

Method:
GET /rest/organization/pricing

Get the current account limitations (pricing) of the organization.

8.3.1 Input parameters

None

8.3.2 Output

Outputs a JSON encoded PricingData object.

8.3.3 Example request

```
GET /rest/organization/pricing
```

8.3.4 Example response

```
HTTP/1.0 200 OK
{"maxNumberOfUsers":10,"maxNumberOfAdmins":2,"daysOfActivityLog":
1,"maxNumberOfDevicesPerUser":20}
```

Activity Management API

9.1 Get activity log

Method:
GET /rest/activity

Returns list of second factor authentication log entries.

9.1.1 Input parameters

The parameters must be provided in the URL query string as URL-encoded. Some of the query parameters are the same as DataTables table plug-in jQuery uses for easier integration purposes. The list of the query parameter keys:

Query parameter key		
length	The maximum number of records to be returned.	optional Default 1000
start	Offset. The number of records to skip from the result.	optional Default 0
search[value]	The value to be used for filtering records. If this value is a substring of the userId, applicationId, applicationName or progress the record will be included. The value null or empty string means no filtering.	optional Default null
columns[<n>][data]	The name of JSON key used as the <n>th column data source.	optional
order[<n>][column]	Column to which ordering should be applied. This is an index reference to the columns array (columns[<n>][data]) of information that is also	optional

	submitted to the server.	
order[<n>][columnName]	Column to which ordering should be applied. This is the name of the column's data source.	optional
order[<n>][dir]	Ordering direction for this column. It will be asc or desc to indicate ascending ordering or descending ordering, respectively.	optional Default: asc

9.1.2 Output

A JSON encoded ActivityList object. (See 10.1)

9.1.3 Example request

Example 1: Query the first 1000 records.

```
GET /rest/activity
```

Example 2: Query the second five records where the userId contains the string "user10" and order the result by columns startedAt ascending and progress descending.

```
GET
/rest/activity?columns[0][data]=userId&columns[1][data]=startedAt
&columns[2][data]=endedAt&columns[3][data]=applicationId&columns[
4][data]=applicationName&columns[5][data]=progress&order[0][colum
n]=1&order[0][dir]=asc&order[1][column]=5&order[1][dir]=desc&star
t=5&length=5&search[value]=user10
```

URL encoded:

```
GET
/rest/activity?columns%5B0%5D%5Bdata%5D=userId&columns%5B1%5D%5Bd
ata%5D=startedAt&columns%5B2%5D%5Bdata%5D=endedAt&columns%5B3%5D%
5Bdata%5D=applicationId&columns%5B4%5D%5Bdata%5D=applicationName&
columns%5B5%5D%5Bdata%5D=progress&order%5B0%5D%5Bcolumn%5D=1&orde
r%5B0%5D%5Bdir%5D=asc&order%5B1%5D%5Bcolumn%5D=5&order%5B1%5D%5Bd
ir%5D=desc&start=5&length=5&search%5Bvalue%5D=user10
```

Example 3: Order the result by userId.

```
https://pbm-dev-pub-
cert.graphi.intra.microsec.hu/register/rest/activity?order[0][col
umnName]=userId
```

9.1.4 Example Response

```
HTTP/1.0 200 OK
{"recordsFiltered":586,"data":[{"startedAt":"2015-02-
05T16:17:12.071138", "endedAt":"2015-02-05T16:17:12.207968",
"validTo":"2015-03-17T11:19:28.183Z", "progress":"DENIED",
"userId":"User100", "applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d",
"applicationName":"application01"},{"startedAt":"2015-02-
05T16:17:12.133678", "endedAt":"2015-02-05T16:17:12.320536",
"validTo":"2015-03-17T11:19:28.183Z", "progress":"APPROVED",
"userId":"User10", "applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d",
"applicationName":"application01"},{"startedAt":"2015-02-
05T16:17:48.643812", "endedAt":"2015-02-05T16:17:48.960763",
"validTo":"2015-03-17T11:19:28.183Z", "progress":"APPROVED",
"userId":"User100", "applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d",
"applicationName":"application01"}],"recordsTotal":18496}
```

Data objects

10.1 ActivityData

An ActivityData object contains the following properties:

userId	The alias of the users current representation.	string
startedAt	Date and time of begin second factor authentication.	ISO 8601 encoded string representation
endedAt	Date and time of end second factor authentication (optional, it is provided if the authentication state is not PENDING).	ISO 8601 encoded string representation
applicazionId	The ID of the application.	string
applicationName	The name of the application (optional, not provided if the application has been deleted).	string (optional)
progress	The state of the second factor authentication.	"PENDING", "APPROVED", "DENIED" or "TIMEOUT" (string)

10.1.1 JSON Example:

```
{"startedAt":"2015-02-05T16:25:41.726687","endedAt":"2015-02-
05T16:25:41.953946","progress":"APPROVED","userId":"User
01","applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d","applicationName":" application01"}
```

10.2 ActivityList

An ActivityList object contains the following properties:

activities	The list of ActivityData objects. See 10.1 ActivityData.	string
recordsTotal	The number of total records.	ISO 8601 encoded string representation
recordsFiltered	Number of records match the search criteria.	ISO 8601 encoded string representation

10.2.1 JSON Example:


```
{"recordsFiltered":2,"data":[{"startedAt":"2015-02-
05T16:17:49.935300","endedAt":"2015-02-
05T16:17:50.179694","progress":"APPROVED","userId":"User
01","applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d","applicationName":"applicazion01"},{"startedAt":"20
15-02-05T16:25:41.726687","endedAt":"2015-02-
05T16:25:41.953946","progress":"APPROVED","userId":"User
01","applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d","applicationName":"applicazion01"},{"startedAt":"20
15-02-05T16:28:12.713145","endedAt":"2015-02-
05T16:28:15.543468","progress":"APPROVED","userId":"User
01","applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d","applicationName":"applicazion01"},{"startedAt":"20
15-02-05T16:28:52.826161","endedAt":"2015-02-
05T16:33:19.743437","progress":"TIMEOUT","userId":"User
01","applicationId":"a591f8b4-333b-4616-9ce6-
d0ee86817e0d","applicationName":"applicazion01"}],"recordsTotal":
100}
```

10.3 AdministratorData

An AdministratorData object contains the following properties:

userId	The userId (registered e-mail address) of the administrator.	string
fullName	The full name of the administrator.	string
phoneNumber	The phone number of the administrator.	string

10.3.1 JSON Example

```
{"userId":"admin@microsec.hu","fullName":"Admin
name","phoneNumber":"123456"}
```

10.4 ApplicationData

An ApplicationData object contains the following properties:

id	The ID of the application.	string
name	The name of the application.	string
organizationId	The OID of the organization the application belongs to.	string

10.4.1 JSON Example

```
{"id":"aacf6f4d-2f31-44e8-b212-
d1328070e11d","name":"application01","organizationId":"1.3.6.1.4.
1.21528.3.3.2.100"}
```

10.5 EnrollmentData

An EnrollmentData object contains the following properties:

enrollmentId	The unique Id of the enrollment.	string
expireDate	The expiry date of the current enrollment.	ISO 8601 encoded string representation
state	State of the enrollment.	"ACTIVE" or "USED" or "EXPIRED"
sent	Determines whether the enrollment was sent to the user's email.	boolean
deviceConfigUrl	An URL pointing to the device configuration. The URL is sent to the user's phone in a QR code during device enrollment.	URL

10.5.1 JSON Example

```
{"enrollmentId":Ot5pL2mkRVO6LYav,"expireDate":"2014-12-
10T16:00:15.065","state":"ACTIVE", "sent": "false",
"deviceConfigUrl": "https://passbyme.com/
deviceConfig/1.3.6.1.4.1.21528.3.3.2.156/URPvk4EuqVJSUfB3?expireD
ate\u003d2015-04-23T10%3A24%3A22.893Z"}
```

10.6 DeviceData

A DeviceData object contains the following properties:

deviceName	The name of the phone.	string
vendorId	The vendor given unique identifier of the device.	string
customerFullName	The full name of the customer.	string
deviceType	The type of the device.	"ANDROID" or "APPLE" (string)
deactivationPassword	This password can be used to delete the corresponding device via the self management interface.	string

10.6.1 JSON Example

```
{"deviceName": "User 01 phone", "vendorId": "vendorId1",
"customerFullName": "User 01", "deviceType": "ANDROID",
"deactivationPassword":"hwuT38NK99"}
```

10.7 InvitationData

An InvitationData object contains the following properties:

expiryDate	End date of the of the invitation codes validity.	ISO 8601 encoded string representation
code	The invitation code.	string

10.7.1 JSON Example

```
{"expiryDate":"2015-01-
30T03:09:47.073","code":"BUmuaZBat8BkNhtk"}
```

10.8 OrganizationData

An OrganizationData object contains the following properties:

organizationId	The unique identifier of the organization.	string
name	The name of the organization.	string
pricing	The account type of the organization.	"Free", "Standard" or "Premium" (string)
email	The registered contact email of the organization.	string
enrollmentExpirationHours	The expiration of the generated enrollment sheets in hours.	integer
invitationExpirationHours	The expiration of the generated invitations in hours.	integer
confirmationLetterEnabled	Determines whether confirmation letter sending is enabled.	boolean

10.8.1 JSON Example

```
{"organizationId": "1.2.3.4", name: "Test Org.", "pricing" :
"Free", "email", "support@testorg.com",
"enrollmentExpirationHours" : 24, "invitationExpirationHours" :
12, "confirmationLetterEnabled": true}
```

10.9 PricingData

A PricingData object contains the following properties:

maxNumberOfUsers	The maximum number of users allowed to exist in the account.	integer
maxNumberOfAdmin	The maximum number of administrators allowed to exist in the account.	integer
daysOfActivityLog	How many days of the activity log is stored and accessible.	integer
maxNumberOfDevicesPerUser	The maximum number of devices a user can enroll for PassBy[ME].	integer

10.9.1 JSON Example

```
{"maxNumberOfUsers":10,"maxNumberOfAdmins":2,"daysOfActivityLog":
1,"maxNumberOfDevicesPerUser":20}
```

10.10 UserData

A UserData object contains the following properties:

oid	The globally unique identifier of the user.	string
shortOID	The identifier of the user which is unique inside the organization.	string
fullName	The full name of the user.	string
email	The registered email address of the user.	string
phoneNumber	The registered phone number of the user.	string (optional)
userId	The alias of the users current representation.	string (optional)
disabled	True if the user was disabled by an administrator, false otherwise.	boolean

10.10.1 JSON Example:

```
{"oid":"1.3.6.1.4.1.21528.3.3.3.109","shortOID":"3.3.109","fullNa
me":"User 01","email":"user01@email.hu","phoneNumber":"123456",
"disabled":false}
```

10.11 LoginNameData

A LoginNameData object contains the following properties:

oid	The globally unique identifier of the user.	string
shortOID	The identifier of the user which is unique inside the organization.	string
userId	The alias of the users current representation.	string

10.11.1 JSON Example:

```
{"oid":"1.3.6.1.4.1.21528.3.3.3.109", "shortOID":"3.3.109",
"userId":"User01" }
```