Documentation
for the use of the
REST interface
of the
Principal Toolbox

Rick Oosterholt
Version 1.13
25-02-2019

SIMPLY FORTES.

Content

1. Interface changes

- Version 8.5.0
  - */node/update/instance/{id}* is deprecated.
    Use */node/update/instance/**{nodeTypeName}**/{id}* instead.
    See *Update instance*

2. Rest API

This document contains a description of the available REST API calls for the Principal Toolbox application.

2.1. General structure

REST API calls have the following structure:

```
{baseurl}/service/fortesipm/{objecttype}/{crud}/{context}
```

where:

`baseurl`	is the base location where the application is hosted, e.g. https://client.principaltoolbox.com
`objecttype`	folder type name, e.g. *PlanItem*
`crud/action`	*create*, *read*, *update, delete, lease, ...*
`context`	e.g. *instance*, *list*, *descendants*

Examples:

https://client.principaltoolbox.com/service/fortesipm/node/read/descendants

https://client.principaltoolbox.com/service/fortesipm/node/create/instance

2.2. Testing REST calls for dummies

Steps to make up your rest call. For testing REST calls, several tools / plugins exist, e.g. Dev HTTP Client or PostMan (plugins for Chrome)

1. Type the REST URL, e.g. https://client.principaltoolbox.com/service/fortesipm/login
2. Choose the request type, usually GET or POST
3. For most REST calls you need to add headers (in PostMan click the "Headers" button). Now fill in the Header name (left) and value (right).
    a. For a JSON body use: Content-Type = Application/JSON;charset=UTF-8
    b. For all calls except the Login call add the following header: PTB-Rest-Authorization = Basic <TOKEN> (where <TOKEN> is retrieved by a previous login call)
4. For GET calls fill in the URL params (in PostMan click the "URL params" button).
5. For POST calls define the POST parameters. Usually, for a JSON call, you have to specify a JSON body, like this:
    a. Click the "raw" button
    b. Select "JSON" in the selector (instead of Text, HTML or XML)
    c. In the text area below, type your JSON. This may be empty content: "{ }" or for a login POST call:

```
{
"username":"username",
"password":"password"
}
```

6. Click "Send" button. The result will be the response for the applied REST call, see below.

2.3. Important notes

Other information that must be taken into account when using the REST interface of the Principal Toolbox is listed below.

- JSON formatting must comply with RFC7159 (https://tools.ietf.org/html/rfc7159)
- For most calls add the header: Content-Type = Application/JSON;charset=UTF-8
- Request type is mostly POST, sometimes GET
    - For POST calls fill in the message body = {} (empty JSON object) or { *JSON content* }
    - For GET calls fill in the URL parameters.
- To do a successful REST call you have to pass a valid authentication token by specifying in the HTTP request header the following:

  ```
  PTB-Rest-Authorization = Basic XYZ123
  ```

  where XYZ123 is the authentication token, which is returned by a succesful login call.

- When using Java for connecting towards the hosted Principal Toolbox application, the minimum version to be used is JVM 1.7+ to support the 4096-bits SSL key.
- When using XML within the JSON parameters, all XML should be stripped from **tabs** and **new lines** and comments.

2.4. Error handling

When a REST call error occurs a HTTP error code (400) is returned together with a RestStatus object.

RestStatus object:

```
{
        "status": {
                "code": 102,
                "text": "FAILURE"
        },
        "message": "This is an error message"
}
```

RestStatus object for Lease exceptions:

```
{
        "status": {
                "code": 502,
                "text": "Lease Failure"
        },
        "message": "This is an error message",
        "details": {
                "lockedOn": "2012-01-01 12:32",
                "lockedBy": {
                        "personID": 2134,
                        "displayName": "User one"
                },
                "leaseTime": 60
        }
}
```

Possible status codes:

Code	Text
101	SUCCESSFUL
102	FAILURE
501	LEASE SUCCESSFUL
502	LEASE FAILURE

3. Logging in

Before executing any of the REST calls provided by the Principal Toolbox, the caller needs to be authenticated. Therefore, the first REST call needs to be a login which provides a session token that can be used on all subsequent calls.
Use of the token is not explicitly mentioned in the remainder of the document but is to be provided at all times through the PTB-Rest-Authorization header (see chapter 1, section 1.2).

3.1. Login

Login via REST.

Available:

6.0+

REST call:

```
POST {baseurl}/service/fortesipm/login
```

URL parameters:

None

POST parameters:

`username`	Username of the account for the login
`password`	Password of the account for the login

Return value

Authentication token if successful. A valid authentication token should be specified in the HTTP request header as follows for subsequent REST calls *(where XYZ123 is the authentication token)*:

```
PTB-Rest-Authorization = Basic XYZ123
```

Example:

To pass these parameters, json format is expected, for example:

```
{
    "username" : "someUsername",
    "password" : "somePassword"
}
```

If not successful the call returns a HTTP UNAUTHORIZED (403). If successful a HTTP OK (200) is returned containing an authentication token in the header.

Example of a successful login response:

```
{
    "status": {
        "code": 101,
        "text": "SUCCESSFUL"
    },
    "message": "Successful lease",
    "details": {
        "loginMessage":"Logged in",
        "authToken":"NmZmNDM1YTEtZDAyZS00ZWM0LWI4NjgtODhkNDM3MmIzNDAz",
        "username":"someUsername",
        "userDisplayName": "and its display name",
        "userID": 1234,
        "administrator": false
    }
}
```

4. Working with instances

Instances are objects (folders) within the Principal Toolbox application. The REST interface allows to read, create, update and delete such instances.

4.1. Read instance

To read instance data, this call can be used.

Available:

6.0+

REST call:

```
GET {baseurl}/service/fortesipm/node/read/instance/{id}
```

URL parameters:

`id` the node id from which the information needs to be retrieved

Return value

Instance data if successful.

Example:

```
{baseurl}/service/fortesipm/node/read/instance/48826857
```

JSON Response

```
{
    "nodeTypeName": "P2Project",
    "nodeTypeID": 5000,
    "id": 48826857,
    "fields": {
        "Custom1": 0,
        "Custom2": "",
        "Custom3": "2011-12-02",
        "Custom4": false,
        "Custom5": [{
            "value": 1492,
            "displayName": "Test"
        }],
        "Custom6": "Generic";

… etc….
    }
}
```

4.2. Read list

Read list returns all the nodes matched by the listing, system wide. This can especially be used for nodes which are not situated on particular locations, e.g. financial categories and skills.

This service is a post method service which means the data needs to be sent to this service via an HTTP post method call. This service looks for a post parameters called "`viewXml`".

Available:

6.0+

REST call:

```
POST {baseurl}/service/fortesipm/node/read/list/{nodeTypeName}
```

URL parameters:

`nodeTypeName`: the node type name of the desired nodes, e.g. P2Project, PlanItem etc. See B.

POST parameters:

`viewXml`: The view definition to be used for this listing (OPTIONAL; when left blank, nodes are not filtered and all fields of the nodes are returned).

`fieldNames`: The field names of the fields to return (OPTIONAL; when left blank, all fields of the nodes are returned).

`sortFields`: The field names to use for sorting (OPTIONAL; when left blank, nodes are returned in order of creation).

`sortDirections`: When using sortFields, also specify this parameter; it can either be ASC or DESC.

Return value

Data in list if successful.

Example:

Retrieve list of nodes (from a certain context)

```
{baseurl}/service/fortesipm/node/read/list/P2Issue (+POST)
```

With POST body[1]:

```
{"viewXml": "<listing id='0'>
        <filter>
                <Name operator='equals' filterValues='Test issue' />
        </filter>
        <layout>
                <header />
                <row>
                        <cell name='Name' />
                        <cell name='Description' />
                        <cell name='FolderID' />
                </row>
        </layout>
</listing>"}
```

Or (alternative POST body – no filter):

```
{"fieldNames": "Name,Description,FolderID"}
```

JSON Response:

```
    {
        "status": {
            "code": 101,
            "text": "SUCCESSFUL"
        },
        "message": "Successful lease",
        "details":  {
             "type":"array",
             "meta":[{
                  "id":6001,
                  "name":"Issue",
                  "displayName":"Issue",
                  "pluralDisplayName":"Issues",
                  "fields":{
                      "Name":{ "displayName":"Name", "type":"string" },
                      "Description":{ "displayName":"Description", "type":"string" },
                      "FolderID":{ "displayName":"ID", "type":"int" }
                  }
             }],
             "value":[
                  {"Name": "Test issue 1", "Description":"Test desc 1", "FolderID": 123},
                  {"Name": "Test issue 2", "Description":"Test desc 2", "FolderID": 456}
             ]
        }
    }
```

[1] For readability, the XML contains **tabs** and **new lines** but this should not be the case when executing the call.

4.3. Read descendants

To read descendants you can use the same approach as the read list call. The descandants call returns all the nodes matched by the listing within the scope of the specified id (location). The information is returned in a non-hierarchical list.

Available:

7.0+

REST call:

```
POST {baseurl}/service/fortesipm/node/read/descendants/{id}/{type}
```

URL parameters:

`id`: the node id from which the descendant needs to be retrieved

`type`: the node type name of the desired nodes, e.g. P2Project, PlanItem etc. See B.

POST parameters:

`viewXml`: The view definition to be used for this listing (OPTIONAL; when left blank, nodes are not filtered and all fields of the nodes are returned).

`fieldNames`: The field names of the fields to return (OPTIONAL; when left blank, all fields of the nodes are returned).

`sortFields`: The field names to use for sorting (OPTIONAL; when left blank, nodes are returned in order of creation).

`sortDirections`: When using sortFields, also specify this parameter; it can either be ASC or DESC.

Return value

Data from descendants if successful.

4.4. Read recursive

To read a tree structure of certain object types beneath a certain object, this call provides the means to retrieve this information in a nested JSON. The call is similar as read descendants call but the information is returned in a hierarchical structure.

Available:

7.0+

REST call:

```
POST {baseurl}/service/fortesipm/node/read/recursive/{id}/{type}/{relation}
```

URL parameters:

`id`: the node id from which the descendant needs to be retrieved

`type`: the node type name of the desired nodes, e.g. P2Project, PlanItem etc. See B.

`relation`: the relation type name to use when reading the tree recursively: normal, planning etc. z See B.

POST parameters:

`fieldNames`: The field names of the fields to return (OPTIONAL; when left blank, all fields of the nodes are returned).

Return value

Nested data with the objects that match the type and relation recursively. Note that the root (location) is included also independently from the requested node type.

Example:

`{baseurl}/service/fortesipm/ node/read/recursive/1455/PlanItem/planning`

No POST body.

Response:


```
{
    "nodeTypeName": "P2Project",
    "nodeTypeID": 5000,
    "id": 1455,
    "fields": {
        "Custom1": 0,
        "Custom2": "",
        "Custom3": "2011-12-02",
        "Custom4": false,
        "Custom5": [{
            "value": 1423,
            "displayName": "Rick Smith"
        }],
        "Custom6": "Generic"
    },
    "objects": [
        {
            "nodeTypeName": 5603,
            "id": 1598,
            "fields": {}
            "objects: [
                {
                    "nodeTypeName": "PlanItem",
                    "nodeTypeID": 5602,
                    "id": 48826868,
                    "fields": {
                        "Custom1": 0,
                        "Custom2": "",
                        "Custom3": "2011-12-02",
                        "Custom4": false,
                        "Custom5": [{
                            "value": 1492,
                            "displayName": "John Williams"
                        }],
                        "Custom6": "Generic"
                    }
                }
            ]
        }
    ]
}
```

4.5. Update instance

REST call to update instance information. Updates can be done using the id.

Available:

8.5+

REST call:

```
POST {baseurl}/service/fortesipm/node/update/instance/{nodeTypeName}/{id}
```

URL parameters:

`nodeTypeName`:	the node type name of the desired nodes, e.g. P2Project, PlanItem etc. See B.
`id`:	ID of the instance

POST parameters:

Instance information may be supplied but is not mandatory.

Return value

Status of the call (success or failure).

Example:

```
{baseurl}/service/fortesipm/ node/update/instance/PlanItem/1455
```

POST body:

```
{
    "Remarks": "Remarks on this instance",
    "Custom2": "123456"
}
```

4.6. Create instance

REST call to create instance information.

Available:

6.0+

REST call:

```
POST {baseurl}/service/fortesipm/node/create/instance/{parentId}/{type}
```

URL parameters:

`parentId`: the node id where the new instance needs to be created

`type`: the node type name of the desired nodes, e.g. P2Project, PlanItem etc. See B.

⚠ Note that the system does not allow to create every type at any location.

POST parameters:

Instance information is to be supplied:

Return value

Status of the call (success or failure).

Example:

```
{baseurl}/service/fortesipm/ node/create/instance/1234/P2Issue
```

POST body:

```
{
    "Description": "REST call does not respond when connection is lost",
    "Remarks": "Remarks on this issue",
    "Custom2": "123456"
}
```

4.7. Create instances

REST call to create new instances. This API call can only be used for nodes which are ‘system-wide’; e.g. Currencies.

Available:

8.5.3.+

REST call:

```
POST {baseurl}/service/fortesipm/node/create/instances/{type}
```

URL parameters:

`type`: the node type name of the desired nodes, e.g. CurrencyRate

⚠ Note that the system does not allow to create every type at system level. Only use when adviced.

POST parameters:

Instance information is to be supplied:

Return value

Status of the call (success or failure).

Example:

```
{baseurl}/service/fortesipm/ node/create/instances/CurrencyRate
```

POST body:

```
{
    “Startdate”: “2018-01-01”,
    “Currency”: 5678980,
    “Rate”: 123.456
}
```

5. Working with meta data

Meta data provides information about the fields (properties) of objects within the Principal Toolbox. It can be used to interpret information from the instances.

Metadata contains information about:

- node type itself
- fields defined by this node type

This service is a post method service which means the data needs to be sent to this service via an http post method call.

⚠ These REST calls are cached so performing the same call again will read the result from the cache and will not gather the same information again

5.1. Read metadata

Returns the metadata for a node type.

Available:

6.0+

REST call:

```
POST {baseurl}/service/fortesipm/node/read/metadata/{nodeTypeName}
```

URL parameters:

`nodeTypeName`: the node type name of the desired nodes, e.g. P2Project, PlanItem etc. See B.

POST parameters:

```
None
```

Return value

Meta data if successful.

Example:

```
{baseurl}/service/fortesipm/node/read/metadata/P2Issue (POST)
```

JSON Response:

```
{
     "id" : 6001,
     "name" : "Issue" ,
     "displayName" : "Issue" ,
     "pluralDisplayName" : "Issues" ,
     "fields" : {
         "Name" : {
             "displayName" : "Name" ,
             "type" : "string" ,
             "isEditable" : true },
         "Custom2" : {
             "displayName" : "Custom 02" ,
             "type" : "int" ,
             "isCustom" : true ,
             "displayWidth" : 60,
             "isEditable" : true },
         "Custom7" : {
             "displayName" : "Custom 07" ,
             "type" : "string" ,
             "inputType" : "select" ,
             "isCustom" : true ,
```

```
            "displayWidth" : 100,
            "isEditable" : true ,
            "selectvalues" : [{
                "value" : "1" ,
                "displayName" : "one" },
            {
                "value" : "2" ,
                "displayName" : "two" },
            {
                "value" : "3" ,
                "displayName" : "three" }]
        }
    }
}
```

5.2. Read available select values for metadata

This call reads the select values of a dynamic select field. These select values are not present in the response of a normal metadata rest call.

Available:

6.0+

REST call:

```
POST /appurl/service/fortesipm/node/read/metadata/selectvalues/{id}/{fieldName}/{nodeTypeName}
```

URL parameters:

`id`: the id of the 'main level' (e.g. Project, Program, Folder, Organisation Unit)

`fieldName`: the field name for which the selectvalues are retrieved

`nodeTypeName`: the node type name of which to retrieve the selectvalues of (e.g. PlanItem, P2Issue)

POST parameters:

None

Return value

Select value meta data if successful.

Example:

`{baseurl}/service/fortesipm/node/read/metadata/selectvalues/631092/Owner/P2Issue (POST)`

JSON Response:


```
[
        {
        "value": "-1",
        "displayName": "--"
        },
        {
        "value": "51626729",
        "displayName": "Jurgen Kalverboer"
        },
        {
        "value": "3193806",
        "displayName": "Mark Fisscher"
        },
        {
        "value": "51437571",
        "displayName": "Michiel Koster"
        }
]
```

6. Working with entries

Entries are used to represent costs, hours, and ‘value’ information
entries can be read in two fashions:

- list separate entries
- list grouped entries (timeline)

6.1. Read entry list

Read entry list returns all the separate entries matched by the listing using the specified id (location).

This service is a post method service which means the data needs to be sent to this service via an HTTP post method call. This service looks for a post parameters called “`viewXml`”.

Available:

6.0+

REST call:

```
POST {baseurl}/service/fortesipm/entry/read/list/{id}
```

URL parameters:

`id`: the node id from which the entry data is to be retrieved

POST parameters:

`viewXml`: The view definition to be used for this call.

Return value

Data in list if successful.

Example:

Retrieve list of entries (from a certain context)

`{baseurl}/service/fortesipm/entry/read/list/{id} (+POST)`

With POST body[2]:

```
{"viewXml": "<listing id='0'>
    <filter sortfield='Resource,Type,PeriodStartdate' sortdirection='ASC,ASC,ASC'
completelist='true'>
        <ValueType operator='in' filterValues='Available,Allocation' />
        <PeriodStartdate operator="bigger" filterValues="today" />
    </filter>
    <layout>
        <header />
        <row>
            <cell name='Resource' />
            <cell name='Type' />
            <cell name='PeriodStartdate' />
            <cell name='Hours' />
        </row>
    </layout>
</listing>"}
```

JSON Response:

```
{
   "type":"array",
   "meta":[
      {
         "id":100,
         "name":"Entry",
         "displayName":"Entry",
         "pluralDisplayName":"Entries",
         "fields":{
            "Resource":{
               "displayName":"Resource",
               "type":"reference"
            },
            "Type":{
               "displayName":"Type",
               "type":"string",
               "selectvalues":[
                  {
                     "value":"Actual",
                     "displayName":"Actual"
                  },
                  {
                     "value":"Budget",
                     "displayName":"Budget"
                  }
               ]
            },
            "PeriodStartdate":{
               "displayName":"Start Date",
               "type":"date"
            },
            "Hours":{
               "displayName":"Total Hours",
               "type":"double"
            }
         }
      }
   ],
   "value":[
      {
         "FolderID":12345,
         "Resource":[
            {
```

[2] For readability, the XML contains **tabs** and **new lines** but this should not be the case when executing the call.

```
                "value":119051,
                "displayName":"User 1"
            }
         ],
         "Type":"Budget",
         "PeriodStartdate":"2019-01-01",
         "Hours":42
      },
      {
         "FolderID":23456,
         "Resource":[
            {
                "value":119051,
                "displayName":"User 1"
            }
         ],
         "Type":"Budget",
         "PeriodStartdate":"2019-01-01",
         "Hours":56

      },
      {
         "FolderID":34567,
         "Resource":[
            {
                "value":119055,
                "displayName":"User 5"
            }
         ],
         "Type":"Budget",
         "PeriodStartdate":"2019-01-03",
         "Hours":234
      }
   ]
}
```

6.2. Read entry timeline

Read entry timeline summarises individual entries against selected properties and time scales.

This service is a post method service which means the data needs to be sent to this service via an HTTP post method call. This service looks for a post parameters called "`viewXml`".

Available:

6.0+

REST call:

```
POST {baseurl}/service/fortesipm/entry/read/timeline/{id}
```

URL parameters:

`id`: the node id from which the timeline data is to be retrieved

POST parameters:

`viewXml`: The view definition to be used for this call.

Return value

Data in list if successful.

Example:

Retrieve entries in a timeline fashion (from a certain context)

`{baseurl}/service/fortesipm/entry/read/timeline/{id} (+POST)`

With POST body[3]:

```
{"viewXml": "<listing id='0'>
    <filter sortfield='Resource,Type,Startdate' sortdirection='ASC,ASC,ASC' completelist='true'>
        <Type operator='in' filterValues='Budget' />
    </filter>
    <layout>
        <header />
        <row>
            <cell name='Resource' />
            <cell name='Type' />
            <cell name='Startdate' />
            <cell name='Hours' />
        </row>
    </layout>
    <timeline>
        <startdate>today</startdate>
        <intervals>5</intervals>
        <scale>weeks</scale>
    </timeline>
</listing>"}
```

JSON Response:

```
{
   "type":"array",
   "meta":[
      {
         "id":100,
         "name":"Entry",
         "displayName":"Entry",
         "pluralDisplayName":"Entries",
         "fields":{
            "Resource":{
               "displayName":"Resource",
               "type":"reference"
            },
            "ValueType":{
               "displayName":"Type",
               "type":"string",
               "selectvalues":[
                  {
                     "value":"Actual",
                     "displayName":"Actual"
                  },
                  {
                     "value":"Budget",
                     "displayName":"Budget"
                  }
               ]
            },
            "Startdate":{
               "displayName":"Start Date",
               "type":"date"
            },
            "Hours":{
               "displayName":"Total Hours",
               "type":"double"
            }
         }
      }
   ],
   "value":[
      {
```

[3] For readability, the XML contains **tabs** and **new lines** but this should not be the case when executing the call.

```
            "Resource":[
               {
                  "value":119051,
                  "displayName":"User 1"
               }
            ],
            "Type":"Budget",
            "Startdate":"2019-01-01",
            "Hours":98
         },
         {
            "Resource":[
               {
                  "value":119055,
                  "displayName":"User 5"
               }
            ],
            "Type":"Budget",
            "Startdate":"2019-01-03",
            "Hours":234
         }
      ]
}
```


Example:

Retrieve entries in a timeline fashion pivotted (see *doPivoting* layout element attribute)

`{baseurl}/service/fortesipm/entry/read/timeline/{id} (+POST)`

With POST body[4]:

```
{"viewXml": "<listing id='0'>
    <filter sortfield='Resource,Type,Startdate' sortdirection='ASC,ASC,ASC' completelist='true'>
        <Type operator='in' filterValues='Budget' />
    </filter>
    <layout doPivoting='true'>
        <header />
        <row>
            <cell name='Resource' />
            <cell name='Type' />
            <cell name='Startdate' />
            <cell name='Hours' />
        </row>
    </layout>
    <timeline>
        <startdate>today</startdate>
        <intervals>5</intervals>
        <scale>weeks</scale>
    </timeline>
</listing>"}
```

JSON Response:


```
{
   "type":"array",
   "meta":[
      {
         "id":100,
         "name":"Entry",
         "displayName":"Entry",
         "pluralDisplayName":"Entries",
         "fields":{
            "Resource":{
               "displayName":"Resource",
               "type":"reference"
            },
            "ValueType":{
               "displayName":"Type",
               "type":"string",
```


[4] For readability, the XML contains **tabs** and **new lines** but this should not be the case when executing the call.

```
                "selectvalues":[
                    {
                        "value":"Actual",
                        "displayName":"Actual"
                    },
                    {
                        "value":"Budget",
                        "displayName":"Budget"
                    }
                ]
            },
            "PVT2018_12_31":{
                "displayName":"2018-12-31",
                "type":"double"
            },
            "PVT2019_01_07":{
                "displayName":"2019-01-07",
                "type":"double"
            },
            "PVT2019_01_14":{
                "displayName":"2019-01-14",
                "type":"double"
            },
            "PVT2019_01_21":{
                "displayName":"2019-01-21",
                "type":"double"
            },
            "PVT2019_01_28":{
                "displayName":"2019-01-28",
                "type":"double"
            }
        }
    },
    "value":[
        {
            "ValueType":"Budget",
            "PVT2018_12_31": 332,
            "PVT2018_01_07": 0,
            "PVT2018_01_14": 0,
            "PVT2018_01_21": 0,
            "PVT2018_01_28": 0
        }
    ]
}
```

7. Working with documents

Documents within the Principal Toolbox can be added at certain locations (that allow documents). Different types of documents can be added.

⚠ The location at which the document is added refers to document containers; this may be different from the 'functional' location (e.g. project). To retrieve the document container…

⚠ These calls must have the content-type multipart or else the default node calls will be applied (see page and further).

7.1. Add document

Add document via REST to a specific location.

Available:

8.0.3+ ⚠ This REST call replaces the `/file/upload` call that was used before (but only within the application itself).

REST call:

```
POST {baseurl}/service/fortesipm/node/create/instance/{id}/Document
```

URL parameters:

`Id` ID for the location where the document (link) is to be added

POST parameters:

The POST data need to be a content-type multipart with parameters (part 1) in JSON and (if applicable) the file content as additional part.

Parameter part:

`fileType` The type of file to be added, should be a MIME types (see table below / next page) or one of these:

FileType	Purpose	Additional Fields
`internaldocument`	Reference to a document already in the toolbox	`InternalLinkID`: ID of the document being referenced
`externaldocumentlink`	External REST service to be called when selected	`ExternalDocumentLink`: Link of the external REST call
`hyperlink`	Generic hyperlink	`Url`: Link to the external document

`Name` Display name of the file

Additional part:

`Content` UU Encoded document content

⚠ This part needs to be provided as form data, not JSON!

File	FileType (MIME types)	Additional Fields
.doc	`application/msword`	`ActualFileName`: Name of the file
.docx	`application/vnd.openxmlformats-officedocument.wordprocessingml.document`	
.xls	`application/vnd.ms-excel`	
.xlsx	`application/vnd.openxmlformats-officedocument.spreadsheetml.sheet`	
.ppt	`application/vnd.ms-powerpoint`	
.pptx	`application/vnd.openxmlformats-officedocument.presentationml.presentation`	
default	`application/octet-stream`	

Table: File types based on MIME type

Return value

Status of the call (success or failure).

Example:

When adding an external document link to the container with the ID: 1234567 with the external document link being 'www.google.com' the following rest call can be made:

```
{baseurl}/service/fortesipm/node/create/instance/1234567/Document
```

POST data (type = form-data):
Parameters (type=text):

```
{
"FileType": "externaldocumentlink",
"Name": "External REST call trigger"
"ExternalDocumentLink": "www.google.com",
}
```

File (type = File):
None

When adding "someDocument.doc" from your system to the same container the following rest call can be made:

```
{baseurl}/service/fortesipm/node/create/instance/1234567/Document
```

POST data (type = form-data):
Parameters (type=text):

```
{
"FileType": "application/msword",
"ActualFileName": "someDocument.doc",
"Name": "someDocument.doc"
}
```

File (type = File):

```
UDGHH23EUA88911AA …. etc.
```

When adding a normal hyperlink to the container with the ID: 1234567, the following rest call can be made:

```
{baseurl}/service/fortesipm/node/create/instance/1234567/Document
```

POST data (type = form-data):
Parameters (type=text):

```
{
"FileType": "hyperlink",
"Name": "Example Link"
"Url": "http://www.example.com/link?par1=test",
}
```

File (type = File):
None

7.2. Update document

Update an existing document (or link) via REST interface.

Available:

8.0.3+

REST call:

```
POST {baseurl}/service/fortesipm/node/update/instance/{id}
```

URL parameters:

`Id`	ID of the document that is to be updated.

POST parameters:

The POST data need to be a content-type multipart with parameters (part 1) in JSON and (if applicable) the file content as additional part.

Parameter part:

`fileType`	The type of file to be added, should be a MIME types (see previous REST call to add document).
`Name`	Display name of the file.

Additional part:

`Content`	UU Encoded document content. ⚠ This part needs to be provided as form data, not JSON!

Return value

Status of the call (success or failure).

Example:

Examples are almost exactly the same as for the create document calls except for the REST call link itself.

8. Working with messages

Messaging allows to set messages on locations within the Principal Toolbox for users to read. Messages can be added through the REST interface as well.

8.1. Send notification message

Sends a message on the given location. The current user is added to the mentions (@).

Available:

7.0+

REST call:

```
POST {baseurl}/service/fortesipm/message/send/{id}
```

URL parameters:

`id` the node id for the message is sent

POST parameters:

`message`	Message to be sent. May contain HTML, including links (a href) etc.
`mentions`	User id's (resource id's) that are referred to (and for whom the message will | appear at their home page).
`hashtags`	Node id's (folder ID's) that are referred to (and where the message will appear).

Return value

Status of the call (success or failure).

Example:

```
{baseurl}/service/fortesipm/message/send/106907
```

POST body:

```
{
        "message" : "<p>Test message that may contain HTML</p>",
        "mentions" : ["8129701",”221345”],
        "hashtags" : ["563655"]
}
```

9. Working with import and export tasks

Import and export tasks within the Principal Toolbox can be executed through the REST interface. The calls allow to ship (or retrieve) the data.

⚠ Tasks need to be configured manually before using them through the REST interface!

9.1. Execute import task

Start an import task with supplied data.

ℹ Administration privileges are required to execute an import task.
ℹ Import format properties of import task is used (e.g. CSV separator).

⚠ For now only CSV (Comma Separated Values) import format is supported (due to streaming restrictions on the use of XML).

Available:

8.0.1+

REST call:

```
POST {baseurl}/service/fortesipm/integration/import/execute/{id}
```

URL parameters:

`id`	The id of the import task to execute

POST parameters:

`data`	Data to import. Data needs to comply to the general formatting requirements of the Principal Toolbox, see A.

Return value

Status of the call (success or failure).

Example:

```
{baseurl}/service/fortesipm/ integration/import/execute/1234
```

POST body
The POST body should be in the proper import format (CSV). This is not JSON!

```
"column1";"column2"
"value1a";"value1b"
"value2a";"value2b"
```

JSON Response

```
{
    "status": {
        "code": 101,
        "text": "SUCCESSFUL"},
    "message": "Import successful",
    "details": {
        "log": "Task TASKNAME with import starting at 2015-05-12 13:44:24\
INFO: Source file not used because task is executed via REST which supplied the data\
INFO: 1 object(en) geïmporteerd (1 bijgewerkt, 0 aangemaakt).\
Finished at 2015-05-12 13:44:24\ ",
        "errors": false,
        "warnings": false
    }
}
```

JSON Response (if error):

```
{
    "status": {
        "code": 102,
        "text": "FAILURE"},
    "message": "Errors ocurred while importing",
    "details": {
        "log": "Task TASKNAME no importmap starting at 2015-05-12 13:47:32\
INFO: Source file not used because task is executed via REST which supplied the data\
ERROR: Import map nog niet gedefinieerd.\
Could not mail task owner because no email address is available.\
Finished at 2015-05-12 13:47:32\ ",
        "errors": true,
        "warnings": false
    }
}
```

9.2. Execute export task

Executes the export task with the given id.

Administration privileges are required to execute an export task.
Export format properties of import task is used (e.g. csv separator).

Available:

8.0.1+

REST call:

```
GET {baseurl}/service/fortesipm/integration/export/execute/{id}
```

URL parameters:

`id` id of the export task to execute.

POST parameters:

None

Return value

Exported data if successful.

Example:

```
{baseurl}/service/fortesipm/ integration/export/execute/1234
```

Response:
Successful exports will result in the data, which is normally exported to a file, being returned to the client.
Faulty requests return a RestStatus object.

successful CSV response:

```
"FolderID";"Description";"ExportDate"
"81748067";"Test description";"2016-08-24"
"105109943";"ANother one";"2016-08-24"
```

failure RestStatus response:

```
{
    "status": {
        "code": 102,
        "text": "FAILURE"
    },
    "message": "Error executing export task",
    "details": {
        "log": "Not a valid export format"
    }
}
```

10. Working with user administration

Users are special entities within the Principal Toolbox which cannot be accessed through the normal 'instance' interface. Moreover, users are a combination of a resource object combined with a login account. As such, the user administration works on both the resource information as well as login account.

⚠ When using (custom) field values to identify users, make sure their value is unique otherwise the REST calls will fail!

ℹ To work with resources only, the instance interface can be utilised.

10.1. Create user

REST call to create a user.

ℹ Administration privileges are required to create a user

Available:

8.0.1+

REST call:

```
POST {baseurl}/service/fortesipm/user/create/instance
```

URL parameters:

None

POST parameters:

`UserName`	Username for the new account
`Lastname`	Last name for the user
`Email`	Email address for the new account

A special variable may be given as well:

`SendEmail`	Whether or not an email should be sent to the new user. To add a comment to the email the special `EmailRemarks` field can be used.

ℹ Provide `ResourceFolderID` in the user details to create a user from an existing resource. The value should be the ID of the existing resource.

⚠ Passwords (`Passwd`) cannot be set via the REST interface and will be removed from the parameters.

Other information may be supplied as well but is not mandatory:

```
isGroup
Initials
Firstname
  …
```

Return value

Complete user information if successful.

Example:

{baseurl}/service/fortesipm/ user/create/instance

POST body:

```
{
    “UserName”: “jansen”,
    “IsGroup”: “false”,
    “Lastname”: “Jansen”,
    “Email”: “jansen@email.com ”,
    "SendEmail": "false"
}
```

JSON Response:


```
{
    "status": {
        "code": 101,
        "text": "SUCCESSFUL"
    },
    "message": "",
    "details": {
        "nodeTypeName": "Person",
        "nodeTypePluralDisplayname": "Resources",
        "nodeTypeDisplayname": "Resource",
        "nodeTypeID": 3,
        "nodeInstanceTypePluralDisplayname": "Resources",
        "nodeInstanceTypeDisplayname": "Resource",
        "id": 129110808,
        "idMainLevel": 101864724,
        "idCurrentOU": -1,
        "mainLevel": {
            "id": 101864724,
            "nodeTypeID": 201,
            "nodeTypeName": "System",
            "members": {
                "0": [],
                "1": [],
                "2": []
            },
            "externalUsers": []
        },
        "securityRoles": [],
        "fields": {
            "Address": "",
            "Description": "",
            "Administrator": false,
            "FolderID": 129110808,
            "HourRate": [],
            "GroupName": "",
            "Gender": "",
            "BusinessUnit": "",
            "Department": "",
            "POBox": "",
            "Company": "",
            "PasswordLastChangedOn": "2015-06-03",
            "ZIP": "",
            "Skill": [],
            "Email": "jansen@email.com",
            "PasswordMustBeChanged": false,
            "City": "",
            "Lastname": "Jansen",
            "Custom2": "",
            "Custom1": 0,
              …
        }
    }
}
```

On failure (user already exists):

```
{
    "status": {
        "code": 102,
        "text": "FAILURE"
    },
    "message": "User already exists",
    "details": null
}
```

Or (missing fields):

```
{
    "status": {
        "code": 102,
        "text": "FAILURE"
    },
    "message": "Missing required field(s): [UserName]",
    "details": null
}
```

10.2. Update user

REST call to update a user details. Updates can be done using the user id or a unique field value as identification.

Administration privileges are required to update user details

Available:

8.0.1+

REST call:

```
POST {baseurl}/service/fortesipm/user/update/instance/{id}
```

Or:

```
POST {baseurl}/service/fortesipm/user/update/instance/{fieldname}/{fieldvalue}
```

URL parameters:

`id`: ID of the user

Or:

`fieldname` A specific fieldname for identification (e.g. Email or a custom field)
`fieldvalue` The corresponding field value

POST parameters:

User information may be supplied but is not mandatory.

```
IsGroup
Initials
Firstname
  …
```

A special variable may be given as well:

`SendEmail` Whether or not an email should be sent to the new user.
To add a comment to the email the special `EmailRemarks` field can be used.

Passwords (`Passwd`) cannot be set via the REST interface and will be removed from the parameters.

Return value

Status of the call (success or failure).

Example:

```
{baseurl}/service/fortesipm/ user/update/instance/Custom1/jsmith98
```

Only changed information needs to be provided.

POST body:

```
{
    "Initials": "J.",
    "Firstname": "Jeremiah",
}
```

10.3. Delete user (inactivate)

REST call to delete a user account. Deletion can be done using the user id or a unique field value as identification.

Administration privileges are required to delete a user

Available:

8.0.1+

REST call:

```
GET {baseurl}/service/fortesipm/user/delete/instance/{id}
```

Or:

```
GET {baseurl}/service/fortesipm/user/delete/instance/{fieldname}/{fieldvalue}
```

URL parameters:

`id`:	ID of the user

Or:

`fieldname`	A specific fieldname for identification (e.g. Email or a custom field)
`fieldvalue`	The corresponding field value

POST parameters:

None

Return value

Status of the call (success or failure).

Example:

```
{baseurl}/service/fortesipm/ user/delete/instance/UserName/jsmith
```

11. Event triggers

The Principal Toolbox allows to configure triggers to external applications on certain events.

HTTPS is only supported for certificates trusted by the JVM/Tomcat.

HTTP authentication is only supported via URL not via POST or HTTP headers. Basic authentication is supported in the following form: http://username:password@host

11.1. Project creation

On successful project creation (on the operational side !) a configured trigger is called.

Available:

8.0.1+

Setting

In order to make use of it, the following system setting must be configured:

```
<Setting>
    <SettingName>CreateProjectWebhookUrl</SettingName>
    <SettingValue>http://mywebhook:8080/something/somethingelse</SettingValue>
</Setting>
```

Request

With this setting, when a project is created an HTTP POST call will be performed to the specified URL with a request body object of the form:

```
{
    "id": 123456
}
```

Where `id` is the id of the created project. Additional information on the project may be requested using the /node/read/instance interface.

Response

The response of the call (trigger) is ignored by the Principal Toolbox but logged in case of a failure (as warning). In case of success, no logging is shown except when enabling the debug flag on the application.

Logging is available to system administrators only (on the server; application log files).

11.2. External document generation

A trigger can be configured for external document generation from within the automated reports section of the Principal Toolbox. The trigger simply defines a link/url to be used from the server to call with the location (object id) as post parameter.

Available:

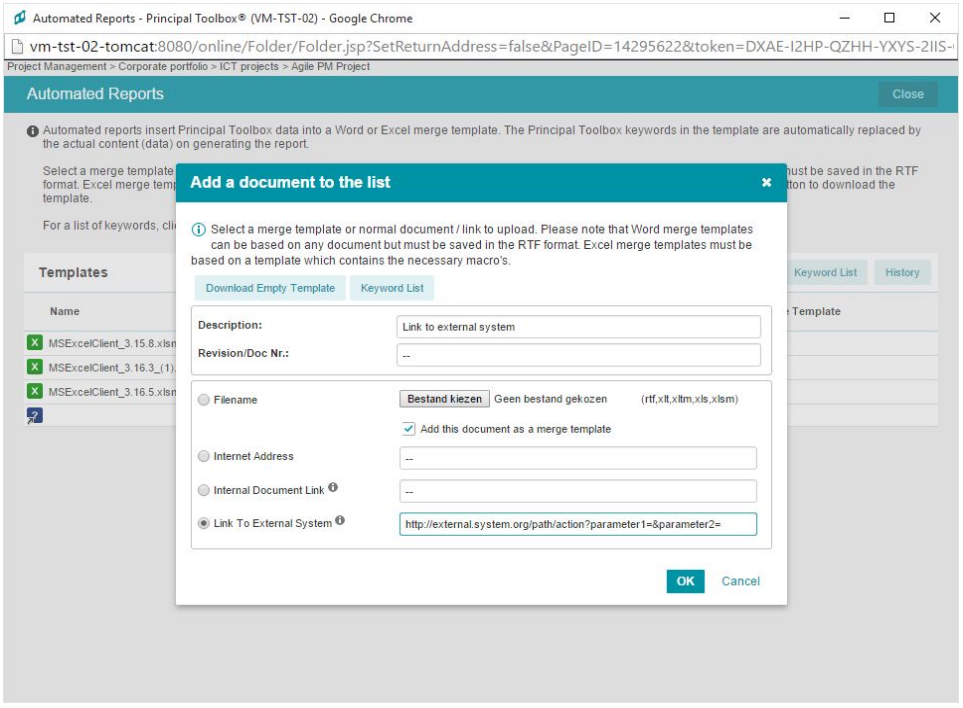
8.0.3+

Setting

In order to make use of it, the following system setting must be configured:

```
<Setting>
    <SettingName>EnableExternalDocumentLinks</SettingName>
    <SettingValue>true</SettingValue>
</Setting>
```

This setting enables the configuration of the mentioned links in automated reports:



Request

When configured, an HTTP POST call will be performed to the specified URL with a request body of the form:

```
{
    "id": 123456
    "personID": 123344
}
```

Where `id` is the id of the location at which the automated report is requested and `personID` is the id of the user that triggered the action. Additional information on this location (e.g. project) may be requested using the /node/read/instance interface.

Response

The response of the call (trigger) is ignored by the Principal Toolbox but logged in case of a failure (as warning). In case of success, no logging is shown except when enabling the debug flag on the application.

Logging is available to system administrators only (on the server; application log files).

A. Data formats

All data exchanged between the Principal Toolbox via the REST interface (and other data interfaces as well) need to comply to following requirements.

Encoding

All data that is exchanged with the Principal Toolbox needs to be encoded using UTF-8.

Dates

Dates need to be specified in the following format:

YYYY-MM-DD

e.g.: `2015-08-04`
`2010-12-31`

Numbers

Numbers need to be specified in the following format:

[-]#[.##]

e.g.: `450`
`2031.40`
`-12.3`

XML

The XML that is exchanged needs to follow general XML syntax guidelines. In addition, the following needs to be taken into account:

- To provide a list of objects (records), the XML needs to have a root tag `<objects> … </objects>`
- Each individual object (record) needs to define it's Principal Toolbox type by the correct tagname, e.g. `<Entry> … </Entry>`
- Within each object (record), the individual field values need to be grouped by the tag `<fields> … </fields>`
- Individual fields need to match the Principal Toolbox internal field names except for import tasks where a mapping between defined field names and the internal field names is applied.

Example XML:

```
<objects>
	<Entry>
		<fields>
			<fasecode>123456.100</fasecode>
			<costsCode >0220051</costsCode >
			<bookDate >2015-01-01</bookDate >
			<Value>5.00</Value>
			<Type>Budget</Type>
		</fields>
	</Entry>
	<Entry>
		<fields>
			<fasecode>123456.100</fasecode>
			<costsCode>0220051</costsCode>
			<bookDate>2015-01-01</bookDate>
			<Value>2.00</Value>
			<Type>Actual</Type>
		</fields>
	</Entry>
</objects>
```

CSV

CSV that is exchanged towards the Principal Toolbox needs to take the following into account:

- As CSV is only used on the import/export, the configured separator (comma, s

- A header row is required to define the field names for each column.
- Headers (field names) cannot contain spaces nor the separator character.
- It is recommended to have all values enclosed within quotes. For string / memo types this is a requirement. Quotes themselves can be escaped using the '\' (slash) character. Newlines can be included within these quotes as well.

Example CSV:

```
ID,name,projectMgr,Text,startDate,someValue
123;"Object 1";A144562;"Description of \"Object 1\" record";2014-12-05;-134.9
133;"Object 2";A154897;"Description of \"Object 2\" record";2015-08-06;-45.2
156;"Object 3";A134221;"Description of
 \"Object 3\" record";2015-02-02;2.9
189;"Object 4";A152312;"Description of \"Object 4\" record";2016-01-01;25.9
```

JSON

The JSON exchanged via the REST interface needs to follow general JSON syntax and follow the specification stated in this document. See also RFC7159 (https://tools.ietf.org/html/rfc7159).

⚠ Note that when mixing XML with JSON (e.g. to specifiy view XML), the XML needs to be stripped from newlines and comments.

B. Node types

Within the Principal Toolbox, many object types exist (e.g. project, plan item, risk etc.). All types can be referenced through the REST interface using their internal name/id. Below is a list of available node types that can be used.

List of node types

Node type			Available	
Name	*ID*	*General name*	*From*[5]	*Until*[6]
Benefit	720	Benefit	7.0+	
BudgetType	8215	Financial category		
CostEntry	83	Classic cost/hour entry		-8.5
CurrencyRate	77	Currency Rates		
DependencyRelation	5700	Dependency		
DiscussionLog	29	Discussion item		
Entry	100	Generic entry	6.0+	-9.5[7]
ExportTask	21	Export task		
History	10	History log		
Idea	4901	Idea	7.0+	
ImportTask	23	Import task		
KnowledgeRepository	4	Documents & knowlegde folder		
Message	8003	Message	7.0+	
NonProjectActivity	5502	Non project activity		
NonProjectActivitySet	5500	Non project activity set		
NonProjectCategory	5501	Non project category		
Order	6061	Order	7.5+	
OrganizationalUnit	500	Organisational unit	8.0+	
P2Action	6041	Action		
P2Change	6031	Change		
P2Issue	6001	Issue		

[5] When left empty, the node type has existed before 6.0.
[6] When left empty, the node type will be supported until further notice.
[7] Since V9.5 Entries are no longer available as node and only retrievable via /entry REST calls

P2Lessonlearned	6051	Lessons learned		
P2Product	5302	Classic project product		-9.0
P2Program	5100	Classic MSP programme folder		-9.0
P2ProgramProject	5110	Classic MSP programme project		-9.0
P2ProgramTemplate	5111	Classic MSP programme model		-9.0
P2Project	5000	Project		
P2ProjectTemplate	5002	Project model		
P2Quality	6021	Quality review		
P2Risk	6011	Risk		
P2Stage	5301	Classic project stage		-9.0
Person	3	User account		
PlanItem	5602	Plan item		
Portfolio	701	Portfolio		
PortfolioDashboard	711	Custom dashboard		
PortfolioModel	703	Portfolio model		
ProgramProject	4910	Programme	8.0+	
ProgramProjectModel	4911	Programme model	8.0+	
ProjectReport	5003	Portfolio item		
ProjectResource	8017	Project resource		
SavedProjectReport	5006	Saved portfolio item		
Scenario	730	Saved portfolio scenario		
Skill	16	Skill		
TimeRegistrationConfiguration	5520	Time entry configuration		
TimeRegistrationGroup	5510	Time entry group		-8.0
Timesheet	81	Time sheet		
TimesheetRow	84	Time sheet row		
URLFolder	400	Link (url)		
UserGroup	30	User group	7.5+	
Workpackage	5303	Classic project work package		-9.0

Normal relations

All nodes in the Principal Toolbox have a 'normal' relation to its parent. The normal relation is used for structuring all data. If a node has no normal relation, the data will not be available and removed on nightly clean up.

Within a project, the nodes are structured as shown below.

P2Project	Project
→ P2Issue	Issue
→ P2Risk	Risk
→ P2Quality	Quality review
→ P2Change	Change
→ P2Action	Action
→ P2Lesson	Lessons learned
→ Order	Order
→ KnowledgeRepository	Document folder
→ Document	Document
→ Entry	Entry

Planning relations

This relation is used within the planning (Gantt). All plan items are hierarchically related to each other the same way as they are shown in the project's planning Gantt.

P2Project
→ PlanItem

Breakdown relations

This relation is used in the product breakdown. All product plan items are hierarchically related to each other the same way as they are shown in the project's product breakdown.

P2Project
→ PlanItem