



# REST API

## Planon Software Suite

Version: L105

© 1997 - 2024 Planon. All rights reserved.

Planon and the Planon logo are registered trademarks of Planon Software Development B.V. or its affiliates. All other product and company names mentioned herein are trademarks or registered trademarks of their respective companies. Planon Software Development B.V., its affiliates and/or licensors own the copyright to all Planon software and its associated data files and user manuals.

Although every effort has been made to ensure this document and the Planon software are accurate, complete and up to date at the time of writing, Planon Software Development B.V. does not accept liability for the consequences of any misinterpretations, errors or omissions.

A customer is authorized to use the Planon software and its associated data files and user manuals within the terms and conditions of the license agreement between customer and the respective legal Planon entity as soon as the respective Planon entity has received due payment for the software license.

Planon Software Development B.V. strictly prohibits the copying of its software, data files, user manuals and training material. However, customers are authorized to make a back-up copy of the original CD-ROMs supplied, which can then be used in the event of data loss or corruption.

No part of this document may be reproduced in any form for any purpose (including photocopying, copying onto microfilm, or storing in any medium by electronic means) without the prior written permission of Planon Software Development B.V. No copies of this document may be published, distributed, or made available to third parties, whether by paper, electronic or other means without Planon Software Development B.V.'s prior written permission.

# About this Document

## Intended Audience

This document is intended for *Planon Software Suite* users.

## Contacting us

If you have any comments or questions regarding this document, please send them to: [support@planonsoftware.com](mailto:support@planonsoftware.com).

## Document Conventions

### **Bold**

Names of menus, options, tabs, fields and buttons are displayed in bold type.

### *Italic text*

Application names are displayed in italics.

### CAPITALS

Names of keys are displayed in upper case.

## Special symbols

	Text preceded by this symbol references additional information or a tip.
	Text preceded by this symbol is intended to alert users about consequences if they carry out a particular action in Planon.

# Table of Contents

Introduction to REST API.....	7
About REST API in general.....	7
Principles of REST API.....	7
REST API Methods.....	8
About the Planon REST API.....	8
Prerequisites.....	10
License.....	10
Navigation panel.....	10
Security.....	11
User access.....	11
Basic features.....	12
User interface.....	12
Definitions level.....	12
Field definitions level.....	13
License usage level.....	14
Which BOs and Fields can be used in REST API calls?.....	15
BOs that are allowed to be used.....	16
Fields that are allowed to be used.....	16
Creating definitions.....	16
Single level definition.....	16
Multi-level definition.....	17
Using definitions.....	19
Generate and import an OpenAPI document.....	19
Creating a sample request.....	20
OpenAPI document.....	23
OPENAPI property.....	23

INFO object.....	23
SERVERS object.....	24
SECURITY object.....	25
PATHS object.....	25
Examples of endpoints.....	26
Create.....	27
Read.....	30
Lookup.....	30
Update.....	31
Delete.....	33
ChangeState.....	35
Retrieve / Download file.....	36
Attach / Upload file.....	38
Detailed information on the body of the request.....	40
Filters in the body.....	40
Values in the body.....	42
Answers in the body.....	43
Dealing with warnings.....	43
Dealing with confirmations.....	44
Arguments in the body.....	46
Advanced features.....	48
File handling.....	48
Remove file.....	48
Attributes on Assets.....	49
Attaching an attribute set.....	50
Detaching an attribute set.....	51
Setting attribute values.....	51
Reading attribute values.....	53

---

Removing attribute values.....	53
Batch processing.....	54
Request header.....	59
Endpoint patterns.....	62
Read endpoint.....	62
Update endpoint.....	63
Delete endpoint.....	65
Execute endpoint.....	66
Lookup endpoint.....	67
ChangeState endpoint.....	68
OpenAPI endpoint.....	69
Batch endpoint.....	73
Download endpoint.....	75
Upload endpoint.....	76
Index.....	78

# Introduction to REST API

REST was chosen as an interface to expose Planon data to other external applications for several reasons. The goal is to offer an Enterprise Application Integration solution to enable integration with external applications.

So, firstly, the solution needs to be:

- An industry standard.
- Secure, light-weight, scalable.
- Able to handle large amounts of data.
- Also, it should not be necessary for customers to implement custom Planon plugins (PaaP Apps) to solve integrations problems.

Security is handled through the standard Planon authentication and authorization functionality. In addition, only Business Objects and fields are exposed that are configured to be exposed in the REST API. As a result, for security reasons, the REST API is not supported on un-authorizable business objects.

## About REST API in general

REST is a software architectural style that describes a uniform interface between physically separate components.

The acronym stands for: **RE**presentational **St**ate **T**ransfer.

## Principles of REST API

There are six basic principles of REST.

### 1. **Stateless**

The requests sent from a client to a server will contain all the required information to make the server understand the requests sent from the client. This can be either a part of URL, query-string parameters, body, or even headers. The URL is used to uniquely identify the resource and the body holds the state of the requesting resource. Once the server processes the request, a response is sent to the client through body, status, or headers.

### 2. **Client-server**

The client-server architecture enables a uniform interface and separates clients from the servers. This enhances the

portability across multiple platforms as well as the scalability of the server components.

### 3. **Uniform Interface**

To obtain the uniformity throughout the application, REST has the following four interface constraints:

- Resource identification
- Resource Manipulation using representations
- Self-descriptive messages
- Hypermedia as the engine of application state

### 4. **Cacheable**

In order to enhance performance, applications are often made cacheable. This is done by labeling the response from the server as cacheable or non-cacheable either implicitly or explicitly. If the response is defined as cacheable, then the client cache can reuse the response data for equivalent responses in the future.

### 5. **Layered system**

The layered system architecture allows an application to be more stable by limiting component behavior. This type of architecture helps in enhancing the application's security as components in each layer cannot interact beyond the next immediate layer they are in. Also, it enables load balancing and provides shared caches for promoting scalability.

### 6. **Code on demand**

This is an optional constraint and which is used the least. It permits a clients code or applets to be downloaded and to be used within the application. In essence, it simplifies the clients by creating a smart application that does not rely on its own code structure.

## REST API Methods

The REST API makes use of the HTTP methods (via a secure connection):

- POST for creating a resource
- GET for retrieving information about a resource
- PUT for updating a resource
- DELETE for deleting a resource

## About the Planon REST API



The Planon application supports REST API, but what does that mean? It allows you to create an interface that is best suited for machine-to-machine communication. The difference between these protocols is that REST API is much more efficient and requires fewer calls to the application.

The Planon REST API only makes use of the POST method. In addition, it uses parameters for the four CRUD operations: Create, Retrieve, Update, and Delete.

The Planon REST API is designed with system-to-system communication in mind.

With the Planon REST API, a user can define the response for an API request based on a Planon business object (BO).

All operations on BOs are triggered by so-called Business Object Methods (BOMs). These BOMs are there to manipulate the BO they belong to. The Planon REST API supports this.

The Planon REST API has three generic endpoints:

- /execute - used for executing BOMs
- /lookup - the standard Planon Lookup functionality
- /changeState - for doing state changes on a BO

To keep more in line with a standard REST API we also created three endpoints for Read, Update and Delete, which are in fact redirects to the /execute endpoint.

- /read
- /update
- /delete
- /upload
- /download

Because there are various Add BOMs in all Planon BOs, it was decided not to create a /**create** endpoint to avoid confusion. If creates are allowed for a BOM, these can be found in the list of methods to execute.



More about endpoints and examples can be found under: [Endpoints](#).

# Prerequisites

To be able to use and work with the REST API, you need to:

- Have a license for it.
- Add the TSI to the navigation panel.
- Arrange security.
- Enable user access.

The following sections describe these in more detail.

## License

The Planon **REST API** license and the Planon **REST API data usage** licenses are needed to be able to work with Planon's REST API.

### Cloud vs. On-Premise

#### Cloud

Since we incur third-party costs for data transfer, Planon Cloud customers who use Planon REST API will be charged for this.

The amount of MBs of all responses is aggregated and stored every 15 minutes. This will be done until no more MBs are left in the license. You can configure alerts to notify if a certain amount of data is consumed or if a threshold is approaching.

Your usage statistics can be found on the **License usage** level of the **REST API** TSI.

#### On-Premise

Since On-Premise customers do not have any data usage, they will be charged for the connector itself.

## Navigation panel

If the REST API has not been added to the GUI yet, you can do that yourself.

1. Go to **Web client > Navigation panel** and add the REST API TSI to your navigation panel. The recommended place to add it, is in the **Tools** navigation group. This step is only required once.
2. Log out & log in.

The REST API can now be configured.

## Security

An access key is needed to access the **REST API** from outside the Planon application. But access keys can only be generated if the Planon software allows us to do that.

Procedure to allow generation of access keys:

1. In the navigation panel, go to **System settings > Security**.
2. Click the **Key pairs** tab
3. The setting **Access key generated?** should have value **Yes**.

If it has value **No**, click on **Generate key pair** in the action panel. That should change it to **Yes**.



If the setting **Access key generated?** was already set to **Yes** and the action is triggered anyways, all existing access keys become invalid.

## User access

If a Planon user needs access to the **REST API** from inside the Planon application, they should be granted REST API authorization by adding them to the user group that holds that authorization. This user group should have access to the navigation group in which the REST API TSI is in (for example Tools).

If a Planon user needs access to the REST API from outside Planon, an **Access key** should be created for them, and the user group they are in should have a **Function profile** that meets the user's needs (for example 'Read').

Procedure to create an access key for a Planon user:

1. In the navigation panel, go to **Accounts > User groups**
2. Go to the **User groups > Users** and select the user for which the access key should be created
3. Go to the **Settings > Access keys**
4. Click **Add**
5. Enter a **Name** and an **Expiry date-time** for the access key.
6. Click **Save**. The access key will be generated.
7. The access key can be copied from the **Access key** field to be used in the calling application.
8. Log out & log in to activate all changes.

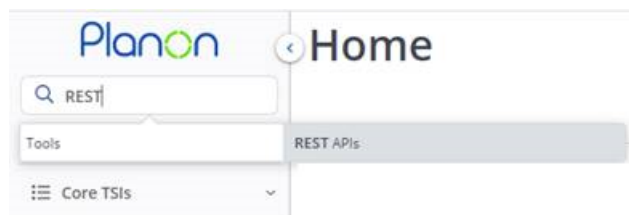
# Basic features

This section describes the user interface of the REST API TSI and its levels and steps.

## User interface

Open the REST API TSI.

You can easily find a TSI by using the Search box in the upper left corner.



The REST API TSI has three levels:

- **Definitions** – to maintain the REST API definitions
- **Field definitions** – to maintain the fields of a REST API definition
- **License usage** – to monitor the hits and MBs that have been used so far

The action panel shows the actions related to the selection level that is active.

## Definitions level

A REST API definition is always based on exactly one BO. Definitions can be interrelated.

The moment a REST API Definition has been created and saved, fields can be added to or removed from it, through **Link field definitions** on the action panel.

Initially, a definition has status **Edit**. All actions are allowed then.

When the definition is ready for use, it can be given status **Published**. Deleting it and editing it is not possible any more then.

The moment a definition should be taken out of service, it can be given status *Unpublished*. If that is meant to be permanent, the unpublished definition can also be deleted. If it was a temporary action, the definition can be given status *Published* again later.

A definition can always be created or copied but can only be updated or deleted when it does not have that status *Published*. When a definition is *Unpublished*, only the name

(**Configuration ID**) and **Comment** field can be changed. When the definition still has status *Edit*, everything can be changed.

---

Field	Description
Business object definition	Select the BO based on which the endpoint should be created. This selection can be done only once.
Configuration ID	<p>The value in this field is automatically generated based on the selected BO earlier. It can be updated as long as the definition does not have status <i>Published</i>.</p> <p>The application does not accept a duplicate configuration ID.</p>
System status	<p>Displays the API definition's status.</p> <p>Initially its status is set to <i>Edit</i>. By clicking the available status transitions in the action panel, the definition can be set to <i>Published</i> and <i>Unpublished</i>.</p>
Comment	In this text field, you can enter notes related to the endpoint definition. This field can be edited at any time.

---

When a *Published* definition needs to be updated, that can be done by making a copy and editing the duplicate definition. The original definition can be *Unpublished* and renamed. Then, the new definition can be given the old name and can be *Published*. Better is to use version numbers on the different versions of your REST API definitions to prevent connected software to fail as a result of updates in the definition.

A group of REST API definitions can be given a specific status all at the same time, by using the **Action on selection** feature. Same goes for the **Delete** action.

You can export one or more REST API definitions, which will result in a zip file that can be saved on your own network. Such an export file can also be **imported** again. Export/import is one of the ways to transport REST API definitions between environments.



You can also transport REST API definitions through [Configuration Transfer](#).

## Field definitions level

Individual fields can be removed from a definition with the **Delete** action as long as the Definition is in Edit mode. This is not allowed any more after the Definition has been published.

The **General** tab of a field definition holds information of the linked BO definition, the audit information, and everything else there is to know about it. If this field is defined as a foreign key in the database, also a reference can be set to the BO definition, which then holds the respective primary key. An example of how to define such a reference can be found in the section [Multi-level definition](#).

Field	Description
System code	Displays the system code of the selected field.
REST API configuration ID	Displays the configuration ID of the selected field.
Business object field definition	Displays the name of the selected field.
Reference as	<p>For reference fields only, indicate whether this field should act as a reference to another business object definition or not:</p> <ul style="list-style-type: none"> <li>• PRIMARYKEY (no reference)</li> <li>• RESTAPIDEFINITION (active reference)</li> <li>• LOOKUP  When using LOOKUP, it is possible to return a description instead of a code.</li> </ul>
Reference to REST API definition	For a reference field - which is a reference to another BO - that business object definition has to be selected here.

## License usage level

There are three selection steps on this level:

- **Hits/MBs today**

Shows the amount of MBs that has been returned to the requester today

- **Daily summary**

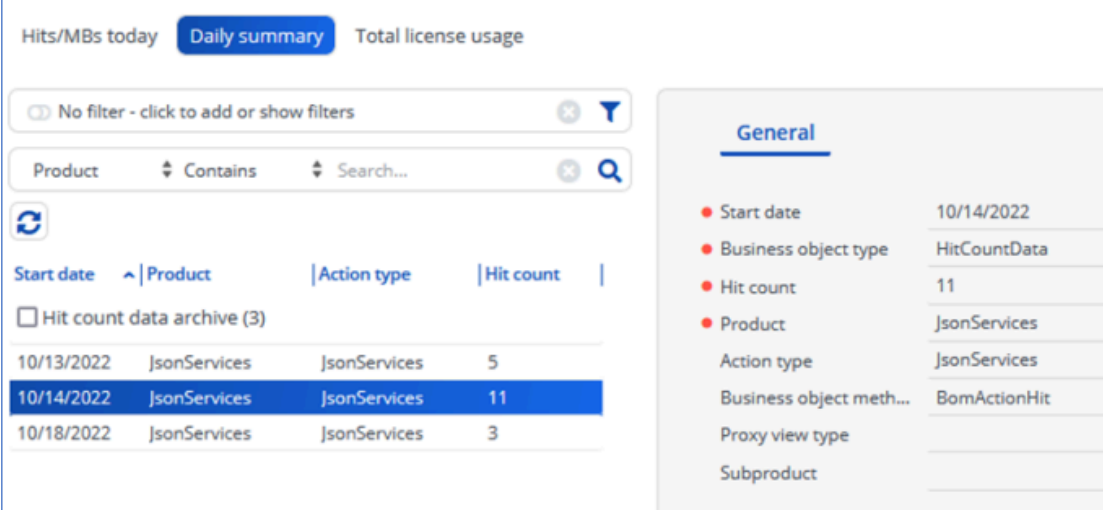
Shows the history of the amount MBs that has been returned to the requester over time

- **Total license usage**

Shows the max usage & total usage, and whether the threshold & max hit counts have been reached or not

License information is updated every 15 minutes.

You can retrieve detailed information about daily usage by clicking the respective line in the list:



Hits/MBs today   **Daily summary**   Total licence usage

No filter - click to add or show filters

Product   Contains   Search...

Start date   Product   Action type   Hit count

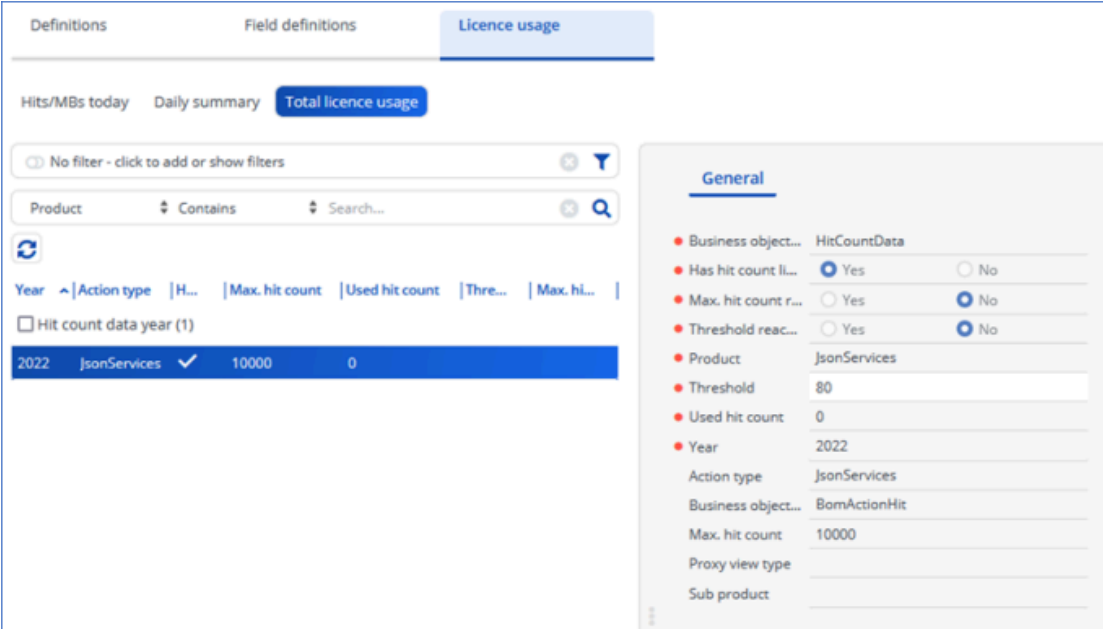
Hit count data archive (3)

Start date	Product	Action type	Hit count
10/13/2022	JsonServices	JsonServices	5
10/14/2022	JsonServices	JsonServices	11
10/18/2022	JsonServices	JsonServices	3

**General**

- Start date: 10/14/2022
- Business object type: HitCountData
- Hit count: 11
- Product: JsonServices
- Action type: JsonServices
- Business object meth...: BomActionHit
- Proxy view type:
- Subproduct:

The **Total licence usage** step also shows the license limits and thresholds:



Definitions   Field definitions   **Licence usage**

Hits/MBs today   Daily summary   **Total licence usage**

No filter - click to add or show filters

Product   Contains   Search...

Year   Action type   H...   Max. hit count   Used hit count   Thre...   Max. hL...

Hit count data year (1)

Year	Action type	H...	Max. hit count	Used hit count	Thre...	Max. hL...
2022	JsonServices	✓	10000	0		

**General**

- Business object...: HitCountData
- Has hit count li...:  Yes    No
- Max. hit count r...:  Yes    No
- Threshold reac...:  Yes    No
- Product: JsonServices
- Threshold: 80
- Used hit count: 0
- Year: 2022
- Action type: JsonServices
- Business object...: BomActionHit
- Max. hit count: 10000
- Proxy view type:
- Sub product:

## Which BOs and Fields can be used in REST API calls?

Not all BOs and not all fields are available for the REST API. This section lists the rules around these selections.

## BOs that are allowed to be used

In general, only BOs that are authorizable, can be exposed through the REST API. There are, of course, a few exceptions.

System BOs are BOs that Planon uses to manage the software. These BOs contain no customer data and are, therefore, not exposed through the REST API.

In addition, deprecated BOs are also not exposed.

## Fields that are allowed to be used

The value of the field should be **In database**. We do not expose calculated fields. Also, the field should be **In use** and **In selection**. You can find these settings in **Field definer**.

Fields that hold passwords and a date-time period are not exposed.

There are more detailed rules on where exactly fields can be used in the body of the request.

For example, if a field is read-only, you cannot use it in the Values section, where the to-be-updated values are listed. More about this can be found in the section on the [OpenAPI document](#).

## Creating definitions

While creating definitions, we distinguish between Single level definitions and Multi-level definitions.

### Single level definition

As already mentioned, REST API definitions can be linked to each other or not. This section explains how to create a not-linked single-level definition.

Procedure to create a REST API definition:

1. Open the **REST API TSI**.
2. Click the **Definitions** level.
3. Click **Add** in the Action panel.

On the **General** tab:

- a. select a **Business object definition** (for example `UsrReservationMeetingRoom`).
- b. optional: edit the **Configuration ID**.
- c. optional: enter a **Comment**.



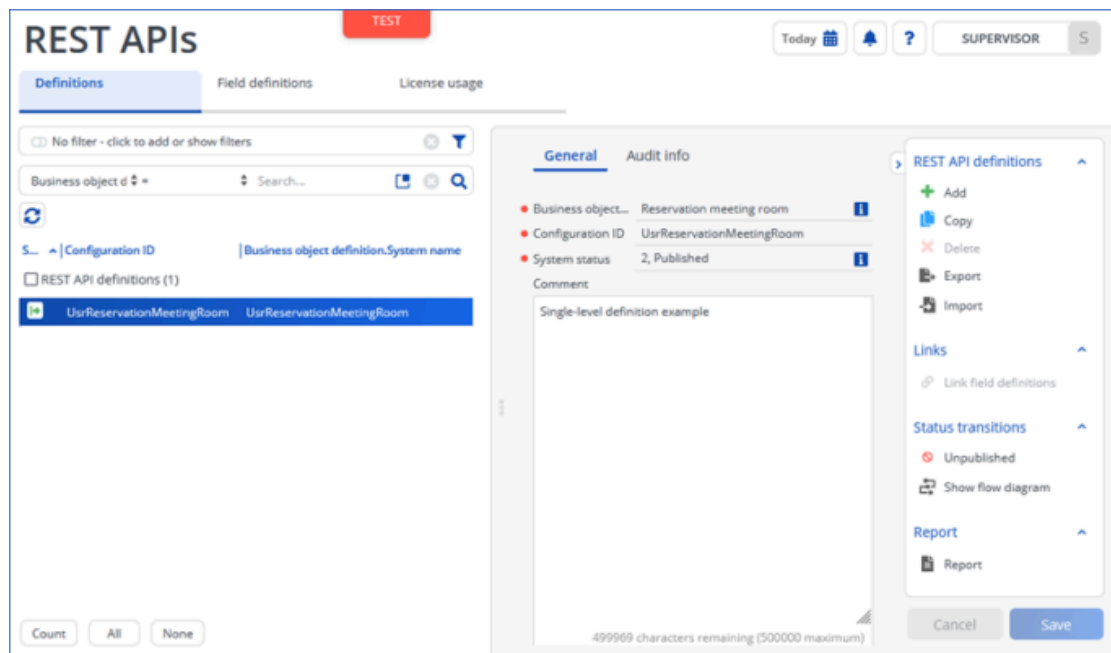
- d. Click **Save**.
  4. Click **Link field definitions** in the Action panel.
 

A dialog box appears, in which fields can be selected by moving them from **Available** to **In use**.
  - a. Make sure that there is at least one field in **In use**. (A REST API definition without fields is pointless and cannot be published).
  - b. Click **OK**.
    5. Click the **Field definitions** level.
 

The fields that you linked in the previous step are listed here.

If required, we can remove individual fields from the definition by selecting them and clicking the **Delete** action in the Action panel.
    6. Navigate back to the **Definitions** level.
    7. Click **Published** in the Action panel.

This REST API definition is now available through its endpoints and, except for the **Comment** field, can no longer be updated.



## Multi-level definition

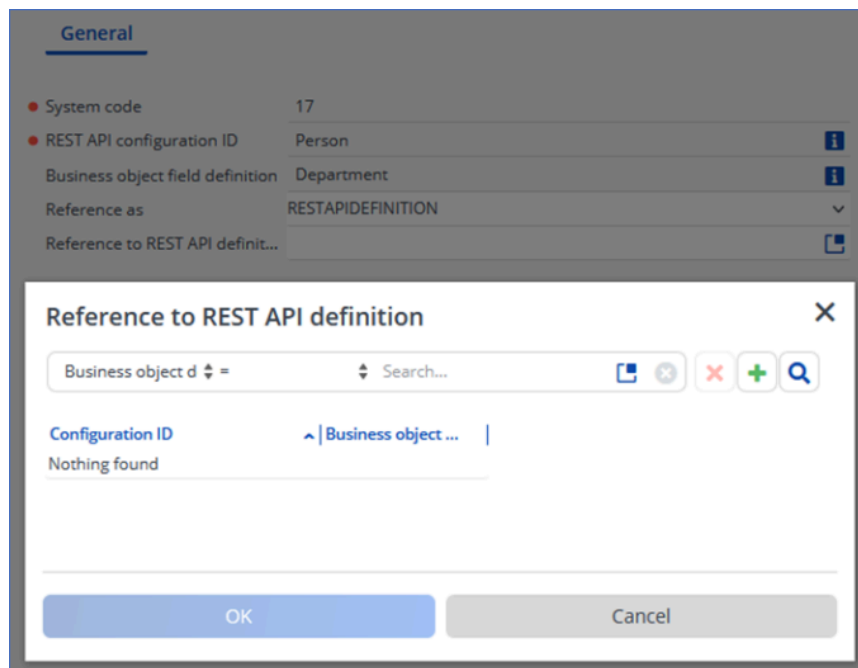
It is also possible to link BOs. For example, linking a Person to the Department where this person works, so that this data can later be retrieved in one call.

In order to make this work, create two single-level definitions as explained in the [previous section](#) (create a definition for **Person** and one for **Department**). Add all field definitions.

These two definitions should be connected: a **Person** works for a **Department**.

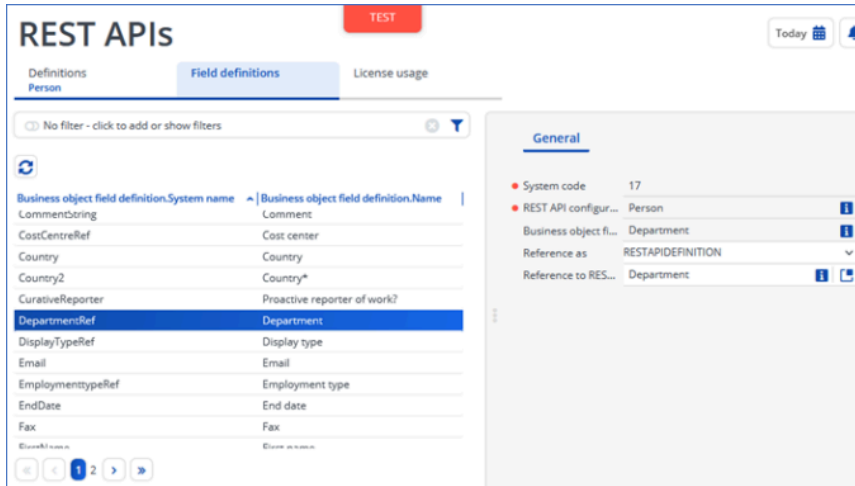
To create this connection:

1. On **Definitions** level: click the **Person** definition
2. On **Field definitions** level: click the **DepartmentRef** field  
This field will become the connection to the Department definition
3. On the **General** tab, this field initially is referenced as a **PRIMARYKEY**. Change that setting to **RESTAPIDEFINITION**.
4. An additional setting becomes available for the **DepartmentRef** field. Click the selection icon. Notice that the **Reference to REST API definition** pop-up does not show any available definitions:



That is, because only definitions that have status *Published* or *Unpublished* are listed here.

5. So, the Department definition needs to be published or unpublished to appear in this list. Please do so.
6. Now, the REST API definition **DepartmentRef** refers to, can be selected. In our case, this is **Department**:



7. Click **Save**
8. Publish both definitions

Remarks:

- You can create as many connections as are necessary.
- Reference fields can be recognized by their name: they usually end with **Ref**.
- You can also create a chain of connections. For example:
  - an **OrderLine** is linked to an **Order**
  - an **Order** is linked to a **Customer**
  - a **Customer** is linked to an **AccountManager** (which is a **Person**)

Note that, for performance reasons, the response of a REST API call only contains the data of one additional level. Deeper levels are ignored.

## Using definitions

In order to verify whether the REST API definitions are defined correctly, an application such as [Postman](#) can be used, which enables you to create and send API requests.

To do this, a few steps need to be taken:

- Generate OpenAPI documents for your REST API definitions
- Import these OpenAPI documents in a tool such as [Postman](#).
- Create new requests in -for example- Postman
- Send your requests to Planon and verify whether the results are as expected

## Generate and import an OpenAPI document

For each REST API definition, an OpenAPI document is available. This document contains information how to use the endpoints for this specific definition and can be downloaded from:

{domain}/sdk/system/rest/v2/openapi/{configuration ID}.json

OpenAPI documents can only be created from REST API definitions that have status *Published*. If they are not *Published*, they cannot yet receive API requests.

Steps to generate an OpenAPI document:

- Suppose the Planon software can be accessed with this url: <https://xxxxx.plnd.cloud/home/BP/WebClient?21>
- And suppose we have published the REST API definition 'Department'
- Then, the OpenAPI document of the Department definition can be accessed through <https://xxxxx.plnd.cloud/sdk/system/rest/v2/openapi/Department.json>
- Save this JSON document on your network
- Import this JSON document in the tool that you will use for sending API requests

Further explanation of the OpenAPI document can be found in section [OpenAPI document](#).

## Creating a sample request

For this example, REST API definitions for the Person and Department BOs should have been created and published. The OpenAPI documents should have been imported in – for example – Postman. If that has not yet been done, go back to [Multi-level definitions](#) and follow the steps from there.

In this sample request, we are looking for a person named John who works for us.

For this request, the /read endpoint should be used.

You can filter on FirstName is John, and EndDate is not filled in.

The complete request then looks like this:

```
{
  "filter": {
    "FirstName": {
      "eq": "John"
    },
    "EndDate": {
      "exists": false
    }
  }
}
```

```
}  
}  
}
```

The answer we receive might contain quite a number of people that have 'John' as their first name. The response will resemble this:

```
{  
  "records": [  
    {  
      "Code": "422",  
      "DepartmentRef": {  
        "Code": "04",  
        "CompositeCode": "04",  
        "IsArchived": false,  
        "Name": "ICT",  
        "SysAccountRef": 1,  
        "Syscode": 10,  
        "SysMutationDateTime": "2020-09-10T00:04:14+01:00",  
        "SysUpdateCount": 0  
      },  
      "Email": "John.Servicedesk@planon.co.uk",  
      "FirstName": "John",  
      "FTEFactor": 1,  
      "IsAnonymized": false,  
      "IsArchived": false,  
      "LastName": "Servicedesk",  
      "MutationDate": "2020-09-09",  
      "PhoneNumber": "+44 (0) 2075016123",  
      "PropertyRef": 3,  
      "Syscode": 149,  
    }  
  ]  
}
```

```

    "SysInsertDateTime": "2020-09-10T00:06:39+01:00",
    "SysMutationDateTime": "2020-09-10T00:32:42+01:00",
    "SysUpdateCount": 2
  },
  {...<data for another person>...
  },
  ...
  {...<data for another person>...
  }
]
}

```

If you know the department John works for, you can reduce the number of results. John works for ICT. As the Person and Department definitions are linked, a nested search can be done. In nested searches, the **syscode** has to be used as the search value. The syscode of the ICT department is 10, so the requests can now be refined into:

```

{
  "filter": {
    "FirstName": {
      "eq": "John"
    },
    "EndDate" : {
      "exists": false
    },
    "DepartmentRef" : {
      "eq": "10"
    }
  }
}

```

This query will reduce the results so that John can more easily be found.  
More examples can be found in the section [Sample usage of the endpoints](#).

## OpenAPI document

To properly document an API, a number of standards are available. Among those, the OpenAPI standard is the most well-known. That is why this standard has been chosen to document the REST API.

The OpenAPI document can be used in tools such as [Postman](#) or [Swagger](#), and is intended for developers to get a better understanding of the workings of the Planon Generic REST API.

Our OpenAPI document consists of the following sections:

- "openapi" section shows the openapi version
- "info" section provides metadata about the API
- "servers" section shows the server where the API is running
- "security" section contains the required security scheme to execute the operations
- "paths" section shows the relative paths to the endpoints and their operations
- "components" section holds a set of reusable objects for various aspects of the OpenAPI Specification

## OPENAPI property

This string must be the semantic version number of the OpenAPI Specification version that the OpenAPI document uses, in our case this version is 3.0.1, and is mandatory.

```
"openapi": "3.0.1",
```

## INFO object

This object provides metadata about the API. The metadata may be used if required and may be presented in editing or documentation generation tools for convenience.

For example:

```
"info": {  
  "title": "Documentation for use of the Planon Generic REST API person definition",
```

```

    "description": "This document describes how to use the Planon Generic REST API person
definition. To use this interface you have to create an api key in the Planon application.",

    "contact": {

        "name": "Market leading Real Estate and Facility Management software",

        "url": "https://planonsoftware.com",

        "email": "info@planonsoftware.com"

    },

    "license": {

        "name": "License with product code E00910 is needed to use the Planon Generic REST
API"

    },

    "version": "2.0.0"

},

```

"title" contains the ConfigurationID of the REST API definition.

"description" contains a short description of the application.

"contact" contains contact information for the exposed API: the name of the organization, the URL that points to the contact information, and the email address of the organization.

"license" contains information about the license that is needed to make use of the Planon Generic REST API.

"version" contains the version number of the document. The version number consists of three parts: major changes, minor changes, and small changes.

## SERVERS object

This object shows the server on which the API is running.

For example:

```

"servers": [

    {

        "url": "https://myenvironment.plnd.cloud/sdk/system/rest/v2"

    }

],

```

The "url"-property contains a URL with the following format:



```
[[DOMAIN]]/sdk/system/rest/v[[API-VERSION-NUMBER]]
```



When a new version becomes available, the old version will be available until it is no longer used.

## SECURITY object

The security object contains the required security scheme to execute the operations.

For example:

```
"security": [  
  {  
    "planon_apikey": []  
  }  
],
```

## PATHS object

This object holds the relative paths to the individual endpoints and their operations. The path is appended to the URL from the server object in order to construct the full URL. The Paths object can contain multiple Path instances.

For non-authorizable BOs, it contains 2 Path instances:

- "/read/{configuration ID}"
- "/lookup/{configuration ID}/{lookup\_value}"

For all other BOs, it also contains a number of additional Path instances:

- "/update/{configuration ID}"
- "/delete/{configuration ID}"
- "/changeState/{configuration ID}" (optional)
- "/execute/{configuration ID}/BomAdd"
- "/execute/{configuration ID}/BomSave"
- and some more, mirroring the BOMs that are exposed by Planon for this specific BO.

Sample Path object:

```

"/lookup/person/{lookup_value}": {
  "post": {
    "description": "Find a record with the given lookup_value as path parameter.
    Uniqueness is not guaranteed, so multiple results could be returned",
    "parameters": [
      {
        "name": "lookup_value",
        "in": "path",
        "description": "Lookup parameter used for field Code to query for the data",
        "required": true,
        "schema": {
          "type": "string"
        }
      }
    ],
    "requestBody": {
      "content": {
        "application/json": {
          "schema": {}
        }
      }
    },
    "responses": {
      "responses": {
        "default": {
          "200": {
            "201": {
              "422": {
                "4XX": {
                  "5XX": {
                    "404": {

```

The POST definition consists of four parts:

- "description" contains a brief explanation of this operation
- "parameters" list the parameters applicable for this PATH (optional)
- "requestBody" allows us to retrieve the body of the request
- "responses" defines what type of responses can be returned
  - **200** when the request was successful
  - **201** when the create request was successful
  - **404** when an incorrect version number has been returned
  - **422** in case of a business error
  - **4XX** in case of a client error
  - **5XX** in case of a server error
  - **Default** otherwise



As this REST API only works on POST, the other options (GET, PUT and DELETE) are not described.

## Examples of endpoints

As explained in [About the Planon REST API](#), six main endpoints are supported:

- {domain}/sdk/system/rest/v2/execute/{configurationID}/{bom}
- {domain}/sdk/system/rest/v2/lookup/{configurationID}/{lookup\_value}
- {domain}/sdk/system/rest/v2/changeState/{configurationID}/  
  {targetState}
- {domain}/sdk/system/rest/v2/read/{configurationID}
- {domain}/sdk/system/rest/v2/update/{configurationID}
- {domain}/sdk/system/rest/v2/delete/{configurationID}

There can be more endpoints, depending on the available BOMs.

The examples in this section, are based on the BO `UsrReservationMeetingRoom`, which allows you to manage meeting room reservations.

## Create

First, let's create a new meeting room reservation 'Demo reservation'. As the BO `UsrReservationMeetingRoom` has one main `BomAdd` method, we will use that one: `/execute/UsrReservationMeetingRoom/BomAdd`.

There are five fields required to be filled in for a reservation:

- Start date & time
- End date & time
- Order group
- Property
- Reservation unit

As order group, property, and reservation unit are references to other BOs, they must contain the unique `syscode` value pointing to a record in their respective BOs.

- 6 refers to order group '05, Project management'
- 8 refers to property '14, Columbus Square'
- 90 refers to reservation unit '0.33 Bach'

The POST request `/execute/UsrReservationMeetingRoom/BomAdd` then looks like this:

```
{
  "values": {
    "BeginDateTime": {
      "date": "2022-12-01",
      "time": "14:00:00"
    }
  },
}
```

```

"EndTime": {
  "date": "2022-12-01",
  "time": "16:00:00"
},
"OrderGroupRef": 6,
"PropertyRef": 8,
"ReservationUnitRef": 90,
>Description": "DEMO reservation"
}
}

```

The response contains many more fields that were filled automatically by the Planon application based on the values given in the request. This is identical to what happens when this request is entered in the Planon application manually:

```

{
  "records": [
    {
      "AllSubOrdersCompleted": false,
      "AppointmentBooking": false,
      "BeginDateTime": {
        "date": "2022-12-01",
        "time": "14:00:00"
      },
      "BeginDateTimeUser": "2022-12-01T14:00:00Z",
      "BusinessObjectDefinitionRef": 730,
      "BusinessObjectStateRef": 229,
      "CumulativeTotalActualCostExclVAT": 25,
      "CumulativeTotalActualCostsInclVAT": 25,
      "Description": "DEMO reservation",
      "DeskConfigurationRef": 20,

```

```
"EHSRequired": false,  
"EndDateTime": {  
    "date": "2022-12-01",  
    "time": "16:00:00"  
},  
"EndDateTimeUser": "2022-12-01T16:00:00Z",  
"EngineerSignOffRequired": false,  
"ExcludeFromInvoiceApproval": false,  
"FloorRef": 123,  
"GroupedInvoice": false,  
"HighPriority": false,  
"IncludeRevenueEstimations": true,  
"InsertDateTime": "2022-12-01T10:58:00Z",  
"InsertSourceSystemRef": 8,  
"Interest": false,  
"InternalCoordinatorPersonRef": 306,  
"LastStateChangedDateTime": "2022-12-01T10:58:00Z",  
"OrderGroupRef": 6,  
"OrderNumber": "266.00",  
"PropertyRef": 8,  
"ProposalStateDateTime": "2022-12-01T10:58:00Z",  
"RefBODefinitionUserDefined": 2918,  
"RefBOStateUserDefined": 1045,  
"ReservationUnitRef": 90,  
"SharedId": "A11B63BD-0C34-4333-9EF3-2E57B44F5E91",  
"SignOffRequired": false,  
"SpaceRef": 262,  
"SysAccountRef": 1,  
"Syscode": 575,  
"SysDataSectionRef": "BP",
```

```
"SysIsArchived": false,
"SysIsDeleted": false,
"SysIsStandardOrder": false,
"SysMutationDateTime": "2022-12-01T10:58:17Z",
"SysOrderID": 266,
"SysUpdateCount": 0,
"TotalActualCostExclVAT": 25,
"TotalActualCostsInclVAT": 25
}
]
}
```

## Read

Let's see if the `/read/UsrReservationMeetingRoom` endpoint can find it:

```
{
  "filter": {
    "Description": {
      "eq": "DEMO reservation"
    }
  }
}
```

Yes! The response is exactly the same as the response we got on the [Create](#).

## Lookup

Using the `/lookup/UsrReservationMeetingRoom/{lookup_value}` endpoint is another way to find one specific reservation.

But which field holds the lookup value? This information can be found in the OpenAPI document in the 'description' field of the 'lookup\_value' parameter of the '/lookup' endpoint:

```
"/lookup/UsrReservationMeetingRoom/{lookup_value)": {
  "post": {
    "description": "Find a record with the given lookup_value as path parameter. Uniqueness is not guaranteed, so multiple results could be returned",
    "parameters": [
      {
        "name": "lookup_value",
        "in": "path",
        "description": "Lookup parameter used for field OrderNumber to query for the data",
        "required": true,
        "schema": {
          "type": "string"
        }
      }
    ],
    "requestBody": {
      "responses": {
      }
    }
  }
},
```

Have a look at the result of the [Create](#) request. The field "OrderNumber" has value "266.00". So, the request should be /lookup/department/266.00.

As the body of the lookup endpoint is always ignored, you can put anything in there:

```
{the quick brown fox jumps over the lazy dog}
```

Best will be:

```
{}
```

The response is exactly the same as what we got with the [Create](#).

## Update

Suppose we want to change the time of the reservation. For that, we use the /update/UsrReservationMeetingRoom endpoint:

```
{
  "filter": {
    "Description": {
      "eq": "DEMO reservation"
    }
  }
},
```

```
"values": {  
  "BeginDateTime": {  
    "date": "2022-12-01",  
    "time": "16:00:00"  
  },  
  "EndDateTime": {  
    "date": "2022-12-01",  
    "time": "17:00:00"  
  }  
}
```

The response shows the updated value:

```
{  
  "records": [  
    {  
      ...  
      "BeginDateTime": {  
        "date": "2022-12-01",  
        "time": "16:00:00"  
      },  
      ...  
      "Description": "DEMO reservation",  
      ...  
      "EndDateTime": {  
        "date": "2022-12-01",  
        "time": "17:00:00"  
      },  
      ...  
    }  
  ]  
}
```



```
}]
```

## Delete

And finally, the reservation can be deleted through `/delete/UsrReservationMeetingRoom`:

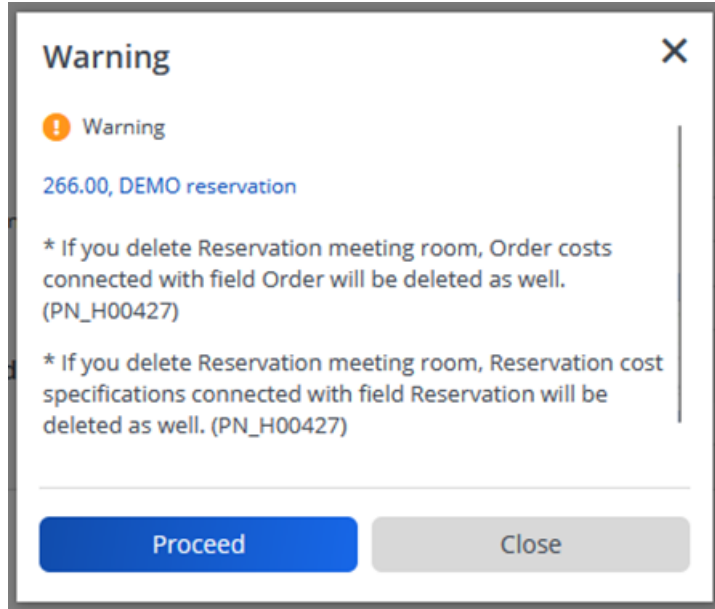
```
{
  "filter": {
    "Description": {
      "eq": "DEMO reservation"
    }
  }
}
```

Note that the response now throws a double warning:

```
{
  "warnings": [
    {
      "code": "PN_H00427",
      "description": "If you delete Reservation meeting room, Order costs connected with field Order will be deleted as well."
    },
    {
      "code": "PN_H00427",
      "description": "If you delete Reservation meeting room, Reservation cost specifications connected with field Reservation will be deleted as well."
    }
  ],
  "confirmations": null,
  "errors": null
}
```

```
}
```

If the same is done directly in the Planon application, you will receive a warning pop-up, after which you can **Proceed**:



These PN\_H00427 warnings can be caught by adding an [Answers block](#):

```
{
  "filter": {
    "Description": {
      "eq": "DEMO reservation"
    }
  },
  "answers": {
    "warnings": [
      {
        "code": "PN_H00427"
      }
    ]
  }
}
```

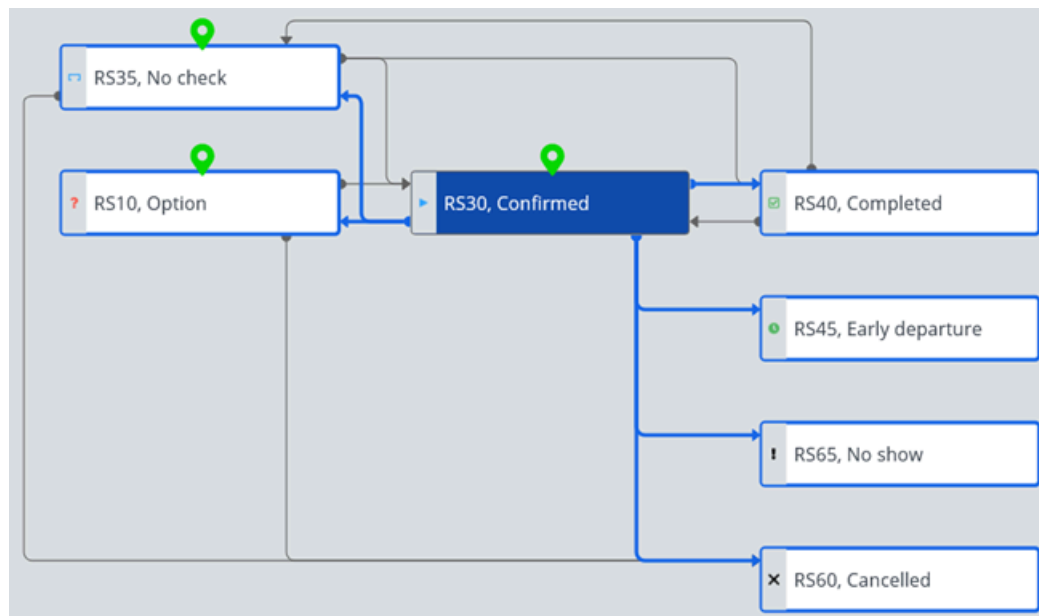
```
}
```

The response shows no records, as the reservation with that name is no longer there:

```
{  
  "records": []  
}
```

## ChangeState

The initial status of a reservation is *Confirmed*. From *Confirmed*, a reservation can go to *No check*, *Option*, *Completed*, *Early departure*, *No show* and *Cancelled* according to the flow diagram in the Planon application:



For example, completing the reservation can be done by calling `/changeState/UsrReservationMeetingRoom/UsrReservationCompleted`:

```
{  
  "filter": {  
    "Description": {  
      "eq": "DEMO reservation"  
    }  
  }  
}
```

```
}  
}
```

The response is exactly the same as what we got with the [Create](#).

## Retrieve / Download file

Downloading a file using the REST API consists of two steps:

1. Get the encrypted path by using the `/read` endpoint
2. Using that encrypted path to retrieve the file

With the `/read` endpoint the encrypted paths of all files and images can be retrieved:

POST `{{baseUrl}}/execute/UsrCommunicationLog/BomRead` Send

Params Auth Headers (12) **Body** Pre-req. Tests Settings Cookies

raw **JSON** Beautify

```

1  [
2  -- "filter": {
3  --- "Name": {
4  --- "eq": "Cleaning matrix Columbus Square"
5  --- }
6  -- }

```

Body 200 OK 154 ms 1.56 KB Save Response

Pretty Raw Preview Visualize **JSON** Copy Search

```

1  [
2  "records": [
3  {
4  "Code": "0052",
5  "DocumentReferral": "LVeI6ChBtawNTopIGEixoMand9KUPuA2j
+AsSZwhwzgirKoksdeixtlPEHaVI0w4Xngj2gRF6edDCy5y
+31moCtGZJPcXjRzBIh6F2b7URagIh4TlySwb84URBdlGzM9D8WWKd7T+8AM
+BD1SfowilhIVtVaF++/fcJ6ycqesRK0UsNEX4QJBYNXHMqDLgu
+JG34xq2hQovoUMBk0adYBqZ7zcYhKYscwNobQzhTtk=/Cleaning Matrix Columbus
Square.docx",
6  "SecureDocumentReferral":
"LVeI6ChBtawNTopIGEixoGTzNaZ0tHdy8ibreoIX2lzDRshjiZkz
+PXhAtJJkeIIftkDantHZxuPHHn/VFTMP4hLcEYFUUWUzqlhL4Kjg0=/Cleaning
Contract.docx",
7  "Syscode": 49,
8  "BeginDate": {
9  "date": "2015-08-01",
10 "time": "09:00:00"
11 },
12 "ImageRef": "LVeI6ChBtawNTopIGEixoD88t9XLyrgM+WcvNRMwXFGGhR2G42Mu
+Qf8v9Jh68eNtKJVB3q6AVeRn7Pj5rnpN3eT5E+u5bGX1eYck/
PAc7qEnF15XjVcTTWUKvFC0cLDzpzNrxHw78PNPXJZNoCQ1w6RkwxRPYtA8FUV3bQ6F3XxFG
/yTbc2dGNvGB3oWNMPTmgm5WeT2qAF0IGwpTUjieA==/Weekly cleaning schedule.
png",
13 "IsArchived": false,
14 "Name": "Cleaning matrix Columbus Square"
15 }
16 ]
17 ]

```

With these answers, the individual files can be downloaded by pasting the encrypted path behind the download path.

For example, the secure document can be downloaded through:

```

GET https://{domain}/sdk/system/rest/v2/download/
LVeI6ChBtawNTopIGEixoGTzNaZ0tHdy8ibreoIX2lzDRshjiZkz
+PXhAtJJkeIIftkDantHZxuPHHn/VFTMP4hLcEYFUUWUzqlhL4Kjg0=/Cleaning Contract.docx

```

## Attach / Upload file

Uploading a file using the REST API consists of two steps:

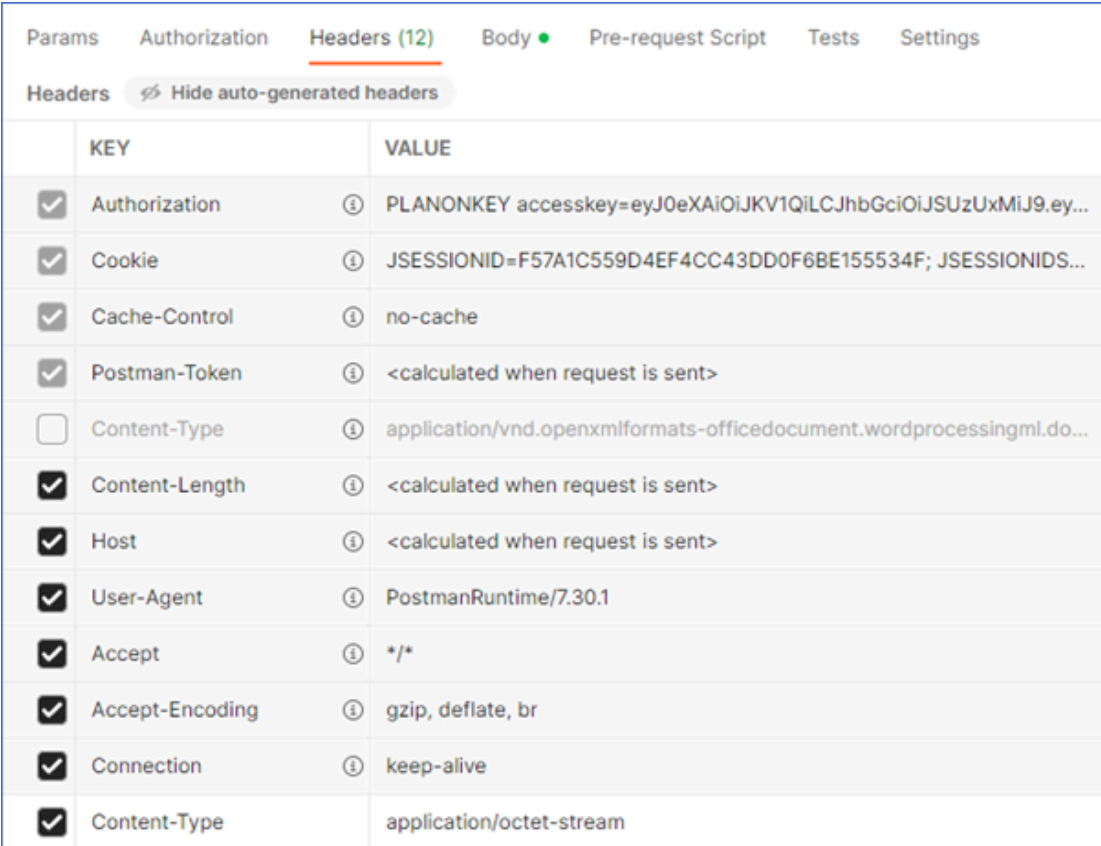
1. Get the UUID by using the /uploadFile endpoint
2. Using that UUID in the /update endpoint

First, we need to create a new endpoint that POSTs to

```
https://{domain}/sdk/system/rest/v2/upload
```

Clear the header key `Content-Type = image/png`

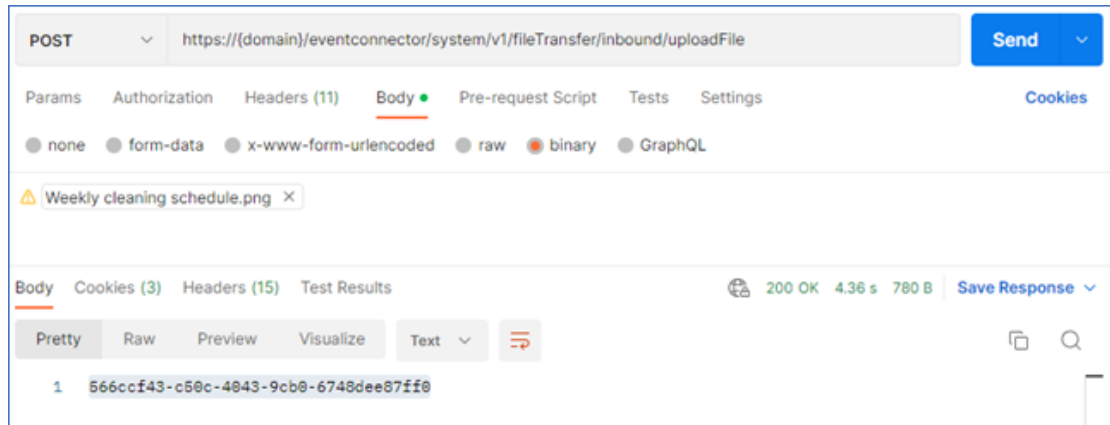
Add a new header key `Content-Type = application/octet-stream`



	KEY	VALUE
<input checked="" type="checkbox"/>	Authorization	PLANONKEY accesskey=eyJ0eXAI0iJKV1QiLCJhbGciOiJSUzUxMiJ9.eyJ...
<input checked="" type="checkbox"/>	Cookie	JSESSIONID=F57A1C559D4EF4CC43DD0F6BE155534F; JSESSIONIDS...
<input checked="" type="checkbox"/>	Cache-Control	no-cache
<input checked="" type="checkbox"/>	Postman-Token	<calculated when request is sent>
<input type="checkbox"/>	Content-Type	application/vnd.openxmlformats-officedocument.wordprocessingml.do...
<input checked="" type="checkbox"/>	Content-Length	<calculated when request is sent>
<input checked="" type="checkbox"/>	Host	<calculated when request is sent>
<input checked="" type="checkbox"/>	User-Agent	PostmanRuntime/7.30.1
<input checked="" type="checkbox"/>	Accept	/*/*
<input checked="" type="checkbox"/>	Accept-Encoding	gzip, deflate, br
<input checked="" type="checkbox"/>	Connection	keep-alive
<input checked="" type="checkbox"/>	Content-Type	application/octet-stream

 The error '415 Unsupported Media Type' pops up, when the additional parameter `Content-Type` with value `application/octet-stream` has not been set.

In the **Body** of the request, you can select a file. A different UUID will be returned for each uploaded file:




The files will be stored in the so-called *Inboundbox*.

Their respective UUIDs can then be used on the `/update` endpoint:

```

1  {
2    "filter": {
3      "Name": {
4        "eq": "Cleaning matrix Columbus Square"
5      }
6    },
7    "answers": {
8      "warnings": [
9        {
10       "code": "PN_H01123"
11     }
12   ]
13 },
14 "values": {
15   "DocumentReferral": {
16     "filename": "Columbus Square.docx",
17     "uuid": "aae5d08e-3cf3-4a78-9f43-539cc7d09a84"
18   },
19   "SecureDocumentReferral": {
20     "filename": "Cleaning contract.docx",
21     "uuid": "a16e48a8-2839-40fc-82a1-d4d8fd1c460a"
22   },
23   "ImageRef": {
24     "filename": "Weekly cleaning schedule",
25     "uuid": "566ccf43-c50c-4043-9cb0-6748dee87ff0"
26   }
27 }
28 }

```

 The warning "The existing file will be overwritten. Do you want to proceed?" can be caught by using the "warning" clause with code "PN\_H01123" in the "answers" section.

The `/update` request will return the updated links to the files:

```

1  |
2  |   "records": [
3  |     {
4  |       "Code": "0052",
5  |       "DocumentReferral": "LveI6ChBtawNTopIGEixoMand9KUPuA2j+AsSZwhwzgzirKoksdeixt1PEHaVI0w4Xngj2gRF6edDCy5y
6  |         +31moCtGZJPcXjRz8Ih6F2b7URagIh4TlySwb84URBdlGzM900wWkd7T+8AM+801SfoWilhIVtVaF+/fcJ6ycqesTFz5jUbJ2
7  |         +9wGygyVgc5n6GpvnNxiEpixzA2ht00F02Q==/Columbus Square.docx",
8  |       "SecureDocumentReferral": "LveI6ChBtawNTopIGEixoGTzNaZ0tHdy8ibreoIX2lzDRshjiZkz
9  |         +PXhAtJJkeIIfTfkDantHZxuPHHn/VFTMP4hLcEYFUUWUzqlhL4Kjg@=/Cleaning Contract.docx",
10 |       "Syscode": 49,
11 |       "BeginDate": {
12 |         "date": "2015-08-01",
13 |         "time": "09:00:00"
14 |       },
15 |       "ImageRef": "LveI6ChBtawNTopIGEixoD88t9XlyrgM+WcvNRmWxFGGhR2G42Mu+Qf8v9Jh6BeNtKJVB3q6AVeRn7Pj5znpN3eT5E
16 |         +u5bGx1eYck/Pac7qEnF15XjVcTTwUKvFC0cLDZpzNzHw78PNPXJZNoCQ1W6RkwxRPYtA8FUV3bQ6F3XxFG/
17 |         yTbc2dGnuGB3oWNPMTmgm5WeT2qAF6IGwpTUjieA==/Weekly cleaning schedule.png",
18 |       "IsArchived": false,
19 |       "Name": "Cleaning matrix Columbus Square"
20 |     }
21 |   ]

```

## Detailed information on the body of the request

### Filters in the body

The filter block in the body is used for filtering the result of a read, lookup or execute call. In the filter, multiple lines can be added, where each line functions as an AND filter.

The following operators are available:

Operator	Description
exists	value = true: returns all records where the given field name has a value.  value = false: returns all records where the given field name does not have a value.
eq	returns all records that match exactly.
ne	returns all records that do not match.
lt	returns all records with a lesser value than the given value.
le	returns all records with a lesser value than or equal to the given value.
gt	returns all records with a greater value than the given value.



---

Operator	Description
ge	returns all records with a greater than or equal to the given value.

---

Note: Use double quotes for strings and no quotes for numbers, true and false.

### Example for Person BO

Looking for a person named 'John' who is working for us.

```
{
  "filter": {
    "FirstName": {
      "eq": "John"
    },
    "EndDate": {
      "exists": false
    }
  }
}
```

### Example for UrsReservationMeetingRoom

Looking for low priority reservations on a specific day for a specific property.

```
{
  "filter": {
    "BeginDateTime": {
      "gt": {
        "date": "2022-12-22",
        "time": "00:00:00"
      }
    }
  }
}
```

```

    }
  },
  "EndDateTime": {
    "It": {
      "date": "2022-12-23",
      "time": "00:00:00"
    }
  },
  "PropertyRef": {
    "eq": 8
  },
  "HighPriority": {
    "eq": false
  }
}
}
}

```

## Values in the body

The values block is used for updating one or more records of a specific business object. Which records should be updated, is defined in the filter block. The to-be-updated values have to be listed in the values block.

### Example

UsrReservationMeetingRoom BO: Updating the start and end time of a specific reservation:

```

{
  "filter": {
    "Description": {
      "eq": "DEMO reservation"
    }
  }
}

```

```

    }
  },
  "values": {
    "BeginDateTime": {
      "date": "2022-12-22",
      "time": "16:00:00"
    },
    "EndDateTime": {
      "date": "2022-12-22",
      "time": "17:00:00"
    }
  }
)
}

```

## Answers in the body

When executing a BOM, warnings can be triggered or confirmations may be required. In order to fully process the request these warnings and confirmations must be bypassed.

This can be done in the **"answers"** block. If no answers are provided, an error will be returned stating the error code, so that the developer can change the request accordingly.

## Dealing with warnings

Warnings can be caught by adding a **"warnings"** clause to the **"answers"** block.

### Example

When deleting a meeting room reservation (/delete/UsrReservationMeetingRoom):

```

{
  "filter": {
    "Description": {
      "eq": "DEMO reservation"
    }
  }
}

```

```
    }  
  },  
  "answers": {  
    "warnings": [  
      {  
        "code": "PN_H00427"  
      }  
    ]  
  }  
}
```

This means that all PN\_H00427 warnings will be ignored in this call. We've already seen in the section on [Delete](#) that removing our reservation threw this same warning twice. Listing it once in the request is good enough.

## Dealing with confirmations

Confirmations can be caught by adding a **"confirmations"** clause to the **"answers"** block.

### Example

When updating the distribution area of a property. In the Planon application, the question comes up whether sub-properties should inherit this change:

### Question ✕

? Question

[36, Columbus Campus](#)

Country: Inherit field has been modified. Do you also want to change data of corresponding non-archived properties? (PN\_A00903)

Yes  No

Distribution area: Inherit field has been modified. Do you also want to change data of corresponding non-archived properties? (PN\_A00903)

Yes  No

City: Inherit field has been modified. Do you also want to change data of corresponding non-archived properties? (PN\_A00903)

Yes  No

---

Proceed
Close

When using a request to do this, these questions can be caught by using an **"answers"** block with a **"confirmations"** clause (/update/Property):


```

{
  "filter": {
    "Name": {
      "eq": "Columbus Campus"
    }
  },
  "answers": {
    "confirmations": [
      {
        "code": "PN_A00903",
        "answer": true
      }
    ]
  }
}

```

```
    ]
  },
  "values": {
    "DistributionArea": "South-West"
  }
}
```

Consequently, all answers to question PN\_A00903 will be set to "Yes" for this call.

 Again, listing it once in the request is good enough.

Note that it is not possible to set some answers to "Yes" and others to "No". Note that when adding multiple "confirmations" clauses for the same confirmation-ID, only the last one will be executed for all.

## Arguments in the body

The arguments block is used when executing a BOM that needs arguments.

In the JSON documentation, all arguments are listed with their respective BOMs. And when arguments are missing in a request, the error message will show their names.

### Example

/execute/UsrAsset/BomDeepCopy: Making a deep copy of an asset with syscode 766.

```
{
  "filter": {
    "Syscode": {
      "eq": 766
    }
  },
  "arguments": {
    "BOMType.AssetMaintenanceChecklistItem": false,
    "BOMType.AssetMaintenanceServicePlan": true,
  }
}
```

```
"BType.CounterMeter": false,  
"BType.DefectListBaseAsset": false,  
"BType.GaugeMeter": false,  
"BType.Hazard": true,  
"BType.InventoryItem": false,  
"BType.InventoryItemComponent": true,  
"BType.MaintenanceActivityDefinitionAdditionalCosts": false,  
"BType.MaintenanceActivityDefinitionManHourCosts": false,  
"BType.MaintenanceActivityDefinitionMaterialCosts": true,  
"BType.StdMaintenanceServicePlanAsset": true,  
"CopyNumber": 1,  
"DestPropertyRef": 6,  
"NrCopies": 1,  
"copyDivergentDataLinkedSrvPlan": false  
}  
}
```

# Advanced features

This section describes more advanced features of using the REST API.

## File handling

It is possible to attach and retrieve documents and images to Business Objects through the REST API.

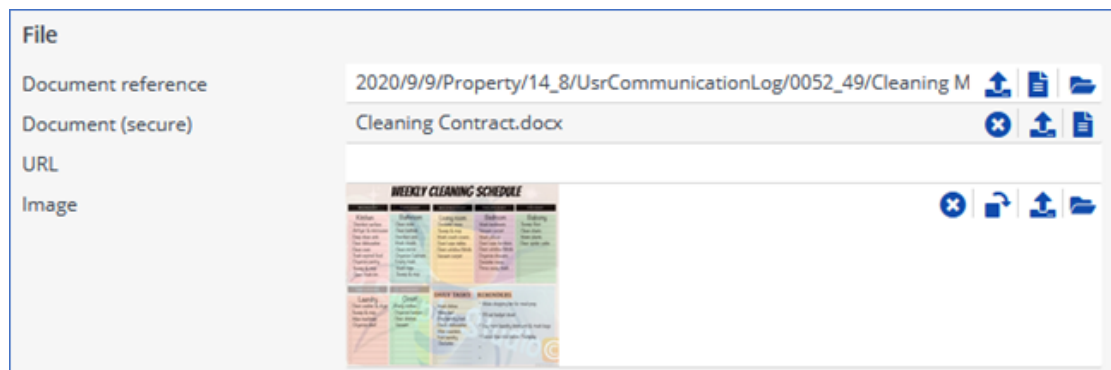
There are two types of file locations:

- regular file locations
- secure file locations

In addition, there are three type of field formats that are relevant here:

- **Document field** – displays the path and the name of the file
- **Secure document field** – displays the name of the file
- **Image field** – shows a preview of the image (jpg, jpeg, png and gif are allowed file types)

In this section, we have a look at the 'Communication logs' of 'Property details'. On the layout, files of all three file types can be added:



The secure document location, allowed file types, and maximum upload sizes are configured in **System settings > File locations**.

## Remove file

For removing files from the Planon application, you can use the /update-request.

In the request, an empty string can be assigned to document fields, which will result in the removal of the file reference.



## Example

The screenshot displays a REST client interface for a POST request to the endpoint `{{baseUrl}}/update/UsrCommunicationLog`. The request body is in JSON format, containing a `filter` object with a `Name` property set to `"Cleaning matrix Columbus Square"`, and a `values` array with three objects: `DocumentReferral`, `SecureDocumentReferral`, and `ImageRef`, each with `filename` and `uuid` properties.

The response is a 200 OK status with a 606 ms response time and 880 B of data. The response body is shown in JSON format, containing a `records` array with one object. This object has the following properties: `Code` (0052), `Syscode` (49), `BeginDate` (a date-time object for 2015-08-01 09:00:00), `IsArchived` (false), and `Name` (Cleaning matrix Columbus Square).

```
POST {{baseUrl}}/update/UsrCommunicationLog

Params Auth Headers (12) Body Pre-req. Tests Settings Cookies
raw JSON Beautify

1
2 --"filter": {
3   --"Name": {
4     --"eq": "Cleaning matrix Columbus Square"
5   }
6 }
7 --"values": [
8   --"DocumentReferral": {
9     --"filename": "",
10    --"uuid": ""
11  },
12  --"SecureDocumentReferral": {
13    --"filename": "",
14    --"uuid": ""
15  },
16  --"ImageRef": {
17    --"filename": "",
18    --"uuid": ""
19  }
20 ]
21

Body 200 OK 606 ms 880 B Save Response
Pretty Raw Preview Visualize JSON

1
2 "records": [
3   {
4     "Code": "0052",
5     "Syscode": 49,
6     "BeginDate": {
7       "date": "2015-08-01",
8       "time": "09:00:00"
9     },
10    "IsArchived": false,
11    "Name": "Cleaning matrix Columbus Square"
12  }
13 ]
14
```

## Attributes on Assets

Unlike SOAP webservices, the REST API can perform read/write actions on the **Asset BO** field **Attributes**.







This section explains how to work with attributes on assets through the REST API.

It is not possible to create an asset, connect it to an attribute set, and set its attribute values all in one go. In the REST API, these action are separate steps.

## Prodedure

1. Create the new asset.
2. Attach an attribute set.
3. Set Attribute values.

In the examples of this section the attribute set **Test attribute-set** will be used. It is connected to these attribute definitions:

Icon	Code	Name
	TASdate	TAS date name
	TASdatetime	TAS date-time name
.01	TASdec	TAS decimal name
12	TASint	TAS integer name
	TASmultiline	TAS multi-line name
	TASdropdown	TAS drop-down name
	TASsingleline	TAS single line name
	TAStime	TAS time name

## Attaching an attribute set

To attach an attribute set to an asset, you can use the `/execute/BaseAsset/ApplyAttributeDefinitionSet`.

```
1  {}
2  .. "filter": {
3  .. .. "Code": {
4  .. .. .. "eq": "000244"
5  .. .. }
6  .. },
7  .. "arguments": {
8  .. .. "AttributeDefinitionSetRef": 23
9  .. }
10 }
```

This request attaches attribute set with syscode=23 to asset with Code="000244".

**i** The arguments AttributeDefinitionSetRef2 and AttributeDefinitionSetRef3 are unavailable for general use until further notice.

To find out what the syscode of the attribute set is, execute the /read/AttributeDefinitionSet request:

```
1  {}
2  .. "filter": {}
3  .. "Name": {}
4  .. "eq": "Test attribute-set"
5  .. }
6  .. }
7  {}
```

## Detaching an attribute set

If, for some reason, the attached attribute set needs to be removed, you can do this by using the /execute/BaseAsset/DetachAttributeDefinitionSet endpoint.

```
1  {}
2  .. "filter": {}
3  .. "Code": {}
4  .. "eq": "000244"
5  .. }
6  .. },
7  .. "arguments": {}
8  .. "AttributeSet1Selected": true
9  .. }
10 {}
```

"AttributeSet1Selected": **true** means that the set will be detached.

"AttributeSet1Selected": **false** means the set will remain attached.


This request detaches attribute set with syscode=23 from asset with code="000244".


**i** The arguments AttributeSet2Selected and AttributeSet3Selected are unavailable for general use until further notice.



## Setting attribute values

After only attaching an attribute set, all its values are still empty.


**Attributes**

Attribute set 1      TestAttSet, Test attribute-set      

TAS date name            

TAS date-time name            

TAS decimal name     

TAS drop-down na...            

TAS integer name     

TAS multi-line name

1000 characters remaining (1000 maximum)

TAS single line name     

TAS time name     

To set the individual attribute values, you can use the `/execute/BaseAsset/BomFieldChange` endpoint:

```

1  |
2  | .. "filter": {
3  |   .. "Code": {
4  |     .. "eq": "000244"
5  |   }
6  | },
7  | .. "values": {
8  |   .. "Attributes": {
9  |     .. "TestAttSet": {
10 |       .. "TASdate": "2023-01-29",
11 |       .. "TASdatetime": "2023-01-29T12:34:00Z",
12 |       .. "TASdec": 12.34,
13 |       .. "TASdropdown": "Option2",
14 |       .. "TASint": 1234,
15 |       .. "TASmultiline": "First line\nSecond line\n\nThird line",
16 |       .. "TASsingleline": "用かほ伸遅エ羽石エエスハ批判シサ庁関協ルせだな青京3馬ゆ善深",
17 |       .. "TAStime": "12:34:00"
18 |     }
19 |   }
20 | }
21 |

```

 `\n` represents a newline character.

## Reading attribute values

For retrieving the attribute values of **Assets** business objects, you can use the `/execute/BaseAsset/BomRead` request.

```
1  |
2  | .."filter":-{}
3  | ..... "Code":-{}
4  | ..... | .."eq":-"000244"
5  | ..... }
6  | .. }
7  | |
```

The attributes are shown as members of the attribute set for the respective business object:

```
1  |
2  | "records": [
3  |   |
4  |   | "AlternativeClassificationRef": 5,
5  |   | "AssetClassificationRef": 2,
6  |   | "AttributeDefinitionSetRef": 23,
7  |   | "Attributes": {
8  |   |   | "TestAttSet": {
9  |   |   |   | "TASdate": "2023-01-29",
10 |   |   |   | "TASdatetime": "2023-01-29T12:34:00Z",
11 |   |   |   | "TASdec": 12.34,
12 |   |   |   | "TASdropdown": "Option2",
13 |   |   |   | "TASint": 1234,
14 |   |   |   | "TASmultiline": "First line\nSecond line\n\nThird line",
15 |   |   |   | "TASsingleline": "用かほ伸運エ羽石エエスハ批判シサ庁關協ルゼだな青京3馬ゆ善深",
16 |   |   |   | "TAStime": "12:34:00"
17 |   |   |   | }
18 |   |   | }
19 |   | "Brand": "Hitachi",
20 |   | "Code": "000244",
21 |   | "ConstructionDate": "2014-01-01".
```

## Removing attribute values

To remove attribute values, you can use `null` for all datatypes in the `/execute/BaseAsset/BomFieldChange` endpoint.

```
1  {}
2  .. "filter": {}
3  ... "Code": {}
4  ..... "eq": "000244"
5  ..... }
6  .. },
7  .. "values": {}
8  ... "Attributes": {}
9  ..... "TestAttSet": {}
10 ..... "TASdate": null,
11 ..... "TASdec": null,
12 ..... "TASdropdown": null,
13 ..... "TASSingleline": null
14 ..... }
15 ... }
16 .. }
17 {}
```

As a result, these four attributes will lose their value. The other attributes in the set keep their value.

## Batch processing

With batch processing a group of requests can be combined into one batch. Each request has a corresponding response showing their id, url and body.

The Microsoft Graph API that we use, has a request limit of 20 individual requests. If that limit is exceeded, an HTTP 429 (Too Many Requests) will be returned.

Error handling:

- Syntax errors cancel the complete batch
- Other errors only cancel the individual request



For now, only the /execute, /delete and /update endpoints have been implemented.

### Sample batch request

When you want to replace the attribute set on an asset, a few steps need to be taken:

1. Remove the current attribute set (if there is any)
2. Attach another attribute set
3. Set the values on the new attribute set

The same example as in the [Attributes on Assets](#) section will be used here.

## Request

```
1  {
2  .... "requests": [
3  ..... { "id": "1", /* Remove the current attribute set */
4  ..... "url": "execute/BaseAsset/DetachAttributeDefinitionSet",
5  > ..... "body": { ...
19 ..... }
20 ..... },
21 ..... {
22 ..... "id": "2", /* Attach another attribute set */
23 ..... "url": "execute/BaseAsset/ApplyAttributeDefinitionSet",
24 > ..... "body": { ...
31 ..... }
32 ..... },
33 ..... {
34 ..... "id": "3", /* Set the values on the new attribute set */
35 ..... "url": "execute/BaseAsset/BomFieldChange",
36 > ..... "body": { ...
54 ..... }
55 ..... }
56 ..... ]
57 }
```

With id 1:

```

{
  "id": "1", /* Remove the current attribute set */
  "url": "execute/BaseAsset/DetachAttributeDefinitionSet",
  "body": {
    "filter": {
      "Code": {"eq": "000244"}
    },
    "arguments": {
      "AttributeSet1Selected": true
    },
    "answers": {
      "confirmations": [
        {
          "code": "PN_H01109",
          "answer": true
        }
      ]
    }
  }
},

```

And id 2:

```

{
  "id": "2", /* Attach another attribute set */
  "url": "execute/BaseAsset/ApplyAttributeDefinitionSet",
  "body": {
    "filter": {
      "Code": {"eq": "000244"}
    },
    "arguments": {
      "AttributeDefinitionSetRef": 3
    }
  }
},

```

And id 3:



```

- {
- "id": "3", /* Set the values on the new attribute set */
- "url": "execute/BaseAsset/BomFieldChange",
- "body": {
-   "filter": {
-     "Code": {"eq": "000244"}
-   },
-   "values": {
-     "Attributes": {
-       "TestAttSet": {
-         "TASdate": "2023-08-29",
-         "TASdatetime": "2023-08-29T12:34:00+01:00",
-         "TASdecimal": "100000",
-         "TASdropdown": "three",
-         "TASinteger": 200000,
-         "TASmultiline": "First line\nSecond line\nThird line",
-         "TASsingleline": "日本文学実地研究レポート",
-         "TAStime": "12:34:00"
-       }
-     }
-   }
- }
- }

```

### Response

- Request id 1 returns an error, as this Asset apparently did not have an attribute set attached yet.
- Request id 2 and 3 are processed correctly.

```
1  |
2  |   "responses": [
3  |     |
4  |     |   "id": 1,
5  |     |   "status": 404,
6  |     |   "body": {
7  |     |     |   "error": {
8  |     |     |     |   "uuid": "bc252b9e-8856-447e-a9f1-3bd317bf08d9",
9  |     |     |     |   "message": "BOM UsrMEAsset.DetachAttributeDefinitionSet is not found."
10 |     |     |   }
11 |     |     }
12 |     |   },
13 |     |   "id": 2,
14 |     |   "status": 200,
15 |     |   "body": { ...
16 |     |   }
17 |     |   },
18 |     |   "id": 3,
19 |     |   "status": 200,
20 |     |   "body": { ...
21 |     |   }
22 |     |   }
23 |   ]
24 | }
```

With id 2:

```
  |
  |   "id": 2,
  |   "status": 200,
  |   "body": {
  |     |   "records": [
  |     |     |   "AttributeDefinitionSetRef": 3,
  |     |     |   "Attributes": {
  |     |     |     |   "TestAttSet": {}
  |     |     |     }
  |     |     |   },
  |     |     |   "Brand": "Hitachi",
  |     |     |   "Code": "000244",
  |     |     |   "ConstructionDate": "2014-01-01",
  |     |     |   "IsPlannedMaintenanceAllowed": true,
  |     |     |   "MaintenanceStartdate": "2016-01-03",
  |     |     |   "Name": "Airco Unit",
  |     |     |   "PhotoRef":
  |     |     |     |   "QmFzZUFzc2V0LzE0NC9QaG90b1JlZi9odHRwOi8vdG9tY2F0LXd1YmRhdjo4MDgxL3dlYmR
  |     |     |     |     |   hdi9JbWFnZXMvVXNyTUUVBc3NldC8wMDAyNDRfMTQ0L0FpcmNvLnBuZw==/Airco.png",
  |     |     |     |   "PropertyRef": 8,
  |     |     |     |   "RequiredConditionScore": "4",
  |     |     |     |   "SpaceRef": 235,
  |     |     |     |   "Syscode": 144,
  |     |     |     |   "Type": "250/40"
  |     |     |   }
  |     |     }
  |     }
  |   }
```

And id 3:

```
{
  "id": 3,
  "status": 200,
  "body": {
    "records": [
      {
        "AttributeDefinitionSetRef": 3,
        "Attributes": {
          "TestAttSet": {
            "TASdate": "2023-08-29",
            "TASdatetime": "2023-08-29T12:34:00+01:00",
            "TASdecimal": "100000",
            "TASdropdown": "three",
            "TASinteger": 200000,
            "TASmultiline": "First line\nSecond line\nThird line",
            "TASsingleline": "日本文学実地研究レポート",
            "TAStime": "12:34:00"
          }
        }
      },
      {
        "Brand": "Hitachi",
        "Code": "000244",
        "ConstructionDate": "2014-01-01",
        "IsPlannedMaintenanceAllowed": true,
        "MaintenanceStartdate": "2016-01-03",
        "Name": "Airco Unit",
        "PhotoRef":
          "QmFzZUFzc2V0LzE0NC9QaG90b1JlZi9odHRwOi8vdG9tY2F0LXd1YmRhdjo4MDgxL3dlYmRhd19JbWFnZXMvVXNyTUUVBc3NldC8wMDAyNDRfMTQ0L0FpcmNvLnBuZw==/Airco.png",
        "PropertyRef": 8,
        "RequiredConditionScore": "4",
        "SpaceRef": 235,
        "Syscode": 144,
        "Type": "250/40"
      }
    ]
  }
}
```

## Request header

Planon uses sessions to store general information that determines the behavior of the application.

This can be information about the language, data section (property set) or reference date. Especially the last option is very important in the area of Spaces & Workspaces.

In order to use this information in REST API, you must add a parameter to the HTTP header. The header itself carries meta information and the parameters carry actual data.

Headers are grouped together alphabetically. If no parameter is set, the response will fall back to the default.

The following table lists the parameter that is currently available to set session data.

Parameter	Contains	Type
ReferenceDate	Sets a sessions reference date.  Note that you should enable the use of reference date to have any effect.	Date

## How it works

If a request is received, a check for custom parameters is performed. All values from custom parameters must be stored.

First, the session is initialized with default settings. The values for these settings are stored in the user profile. Once the session is initialized the stored values from the header parameters must be assigned to the appropriate session properties and only then the request itself can be processed.

## ReferenceDate

You can use a reference date to access time-dependent data. Setting a reference date allows you to retrieve only business objects that are valid either on, before or after this date.

- If the header does not contain the ReferenceDate parameter, the default date (current date) is active and nothing is done.
- If the header contains the ReferenceDate parameter and that has a value, this value needs to be validated:
  - If the value equals *none*, the *useReferenceDate* session parameter is set to *False*. This means that the reference date is not taken into account when filtering business objects.
  - If the value does not equal *none*, the value must be validated for a proper date. This date must comply to the ISO8601-standard, which is *YYYY-MM-DD*. In this example, the *1st of October 2023* would then be *2023-10-01*.
  - If the date does not match this format or the date itself is invalid, the current date stays active.
- If the date passes validation, the *ReferenceDate* session parameter is set to the given date.

## Example

```
ReferenceDate: 2023-10-01
Content-Type: application/json
```

Authorization: PLANONKEY accesskey=eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzUxMiJ9.  
[...r\_3LCaao7RxYuh06Q

User-Agent: PostmanRuntime/7.33.0

Accept: \*/\*

Postman-Token: e6be7525-4749-48e2-9228-1f7d2abcaabe

Host: tbschout-acc.plnd.cloud

Accept-Encoding: gzip, deflate, br

Connection: keep-alive

Content-Length: 68

Cookie: JSESSIONID=FC792A8BF242D76A547339933F0FD6B1;  
PLANONINGRESS=f262276955e44[...]292805c80

# Endpoint patterns

The REST API must support all *CRUD*-operations as well as *Lookup*- and *Statechange*-operations on a selected business object.

It is important to have the *Statechange*-operation, because changing a business object's status may trigger some business logic.

The REST API also needs to support an *Execute*-operation for executing different BOMs on a business object.

Finally, there is an endpoint for *uploading* and for *downloading*.

In addition to these operations, the REST API also needs to support versioning to be able to implement changes without *breaking* a customer's implementation.



In order not to expose too much information in the actual REST API, only POST-requests are used.

The following sections describe the supported endpoints.

## Read endpoint

The *Read*-endpoint is meant to read a business object defined in the endpoint definition. The result can contain more than 1 business object.

### Example

```
http://{environment}/sdk/system/{version}/read/{definition.configurationID}
```

### Request

In the request body you can use the *filter*-tag to select the appropriate business objects.

```
{  
  "filter": {  
    "Syscode": {"eq": 260 }  
  }  
}
```

```
}  
}
```

## Response

The response will be a list of records that apply to the filter.

```
{  
  "records": [  
    {  
      "City": "Boxmeer",  
      "SysMutationDateTime": "2021-09-15T15:14:01+02:00",  
      "Country": "Netherlands",  
      "Syscode": 260  
    }  
  ]  
}
```

## Update endpoint

The *Update*-endpoint is meant to update a business object defined in the endpoint definition.

In the request body there are 4 tags: *filter*, *values*, *answers* and *arguments*. In the *filter* tag you specify the business object you want to update. In the *values* tag you set the values for the properties for that business object.



It is not possible to update multiple business objects in one go. If the filter selects more than one business object, only the first one is processed.

## Example

```
http://{environment}/sdk/system/{version}/update/{definition name}
```

## Request

```
{
  "filter": {
    "Syscode": {"eq": 260 }
  },
  values: {
    "Country": "Netherlands"
  }
}
```

## Response

The response will contain the updated business object and the fields that are defined in the endpoint definition.

```
{
  "records": [
    {
      "City": "Boxmeer",
      "SysMutationDateTime": "2021-09-15T15:14:01+02:00",
      "Country": "Netherlands",
      "Syscode": 260
    }
  ]
}
```



```
}
```

## Delete endpoint

The *Delete*-endpoint is meant to delete a business object that is defined in the endpoint definition.

It is unlikely that this operation will be used often, but it is there to comply to business requirements. In the body of the request, the *filter*-tag can be used to select the appropriate business object.



It is not possible to delete multiple business objects in one go. If the filter selects more than one business object, only the first one is processed.

### Example

```
http://{environment}/sdk/system/{version}/delete/{definition name}
```

### Request

```
{  
  "filter": {  
    "Syscode": {"eq": 260 }  
  }  
}
```

### Response

```
{
```

```
}
```

## Execute endpoint

The *Execute* endpoint is available to trigger the business object's BOMs (methods) from the endpoint definition.

This means that all functionality that is available on a business object can be used through the REST API. If a BOM needs parameters, they are passed in the request body as *arguments*.



It is not possible to execute a BOM on multiple business objects in one go. If the filter selects more than one business object, only the first one is processed.

### Example

```
http://{environment}/sdk/system/{version}/execute/{definition name}/{BOM name}
```

### Request

In the request body the *values* tag sets the values for the properties on the new business object.

```
{
  "values" : {
    "IsUpdateQtyOnNumberOfPersonChange": false,
    "ItemCostsExclVAT": 0.00,
    "ItemCostsInclVAT": 0.00,
    "OrderRef": 540,
    "ReportedBack": false
  }
}
```

## Response

The response will contain the newly created business object and the fields that are defined in the endpoint definition.

```
{
  "records": [
    {
      "IsUpdateQtyOnNumberOfPersonChange": false,
      "ItemCostsExclVAT": 0.00,
      "ItemCostsInclVAT": 0.00,
      "OrderRef": 540,
      "ReportedBack": false
    }
  ]
}
```

## Lookup endpoint

The *Lookup* endpoint is meant to search on certain lookup values that are defined on the business object from the endpoint definition.

The lookup-value is added to the URL and will look as follows:

## Example

```
http://{environment}/sdk/system/{version}/lookup/{definition name}/{lookup value}
```

## Response

The response will contain 1 or more business object that meet the lookup-value.

```
{
  "records": [
    {
      "City": "Boxmeer",
      "SysMutationDateTime": "2021-07-15T15:14:01+02:00",
      "Country": "Netherlands",
      "Syscode": 260
    },
    {
      "City": "Wijchen",
      "SysMutationDateTime": "2021-09-01T11:58:01+02:00",
      "Country": "Netherlands",
      "Syscode": 460
    }
  ]
}
```

## ChangeState endpoint

The *changeState* endpoint is important because business logic is triggered when changing the status of a business object. Consequently, an endpoint must be available.



It is not possible to change a status on multiple business objects in one go. If the filter selects more than one business object, only the first one is processed.

### Example

```
http://{environment}/sdk/system/{version}/changeState/{definition name}/{target state}
```

## Request

In the request body you can use the *filter* tag to select the appropriate business object.

```
{
  "filter": {
    "Syscode": {"eq": 260 }
  }
}
```

## Response

The response will contain the updated business object and the fields that are defined in the endpoint definition.

```
{
  "records": [
    {
      "City": "Boxmeer",
      "SysMutationDateTime": "2021-09-15T15:14:01+02:00",
      "Country": "Netherlands",
      "Syscode": 260
    }
  ]
}
```

## OpenAPI endpoint

To properly document an API, a number of standards are available. The OpenAPI standard is the best known standard, which is why it has been selected to document the REST API.

The OpenAPI document can be used in tools such as [Postman](#) or in [Swagger](#), and is intended for developers to get a better understanding of how the Planon Generic REST API works.

## Example

```
http://{environment}/sdk/system/{version}/openapi/{definition name}.json
```

Calling this endpoint will return a JSON file for the definition mentioned in the endpoint.

## Response

```
{
  "openapi": "3.0.1",
  "info": {
    "title": "Documentation for use of the Planon Generic REST API Department definition",
    "description": "This document describes how to use the Planon Generic REST API Department definition. To use this interface you have to create an api key in the Planon application.",
    "contact": {
      "name": "Market leading Real Estate and Facility Management software",
      "url": "https://planonsoftware.com",
      "email": "info@planonsoftware.com"
    },
    "license": {
      "name": "License with product code E00910 is needed to use the Planon Generic REST API"
    },
    "version": "2.0.0"
  }
}
```

```

},
"servers": [
  {
    "url": "https://planon-acc.plnd.cloud/sdk/system/rest/v2"
  }
],
"security": [
  {
    "planon_apikey": []
  }
],
"paths": {
  "/read/Department": {
    "post": {
      "description": "Read data",
      "requestBody": {
        "content": {
          "application/json": {
            "schema": {
              "properties": {
                "filter": {
                  "$ref": "#/components/schemas/filter"
                }
              }
            }
          }
        }
      },
      "required": true
    },
    "responses": {

```

```

"422": {
  "description": "Returned upon a business error",
  "content": {
    "application/json": {
      "schema": {
        "$ref": "#/components/schemas/businessMessages"
      }
    }
  }
},
"4XX": {
  "description": "Returned upon a client error",
  "content": {
    "application/json": {
      "schema": {
        "$ref": "#/components/schemas/systemError"
      }
    }
  }
},
[...],
},
"securitySchemes": {
  "planon_apikey": {
    "type": "apiKey",
    "description": "Use for value: 'PLANONKEY acceskey={{SEC-TOKEN}}'",
    "name": "Authorization",
    "in": "header"
  }
}
}

```



```
}  
}
```

## Batch endpoint

The *batch*-endpoint provides a way to send up to 20 requests in one go. Each request will have a separate response inside the response and is identified by the *id* set in the request.

### Example

```
http://{environment}/sdk/system/{version}/batch
```

### Request

In the body of each request, the *filter*-tag can be used to select the appropriate business objects.

```
{  
  "requests": [  
    {  
      "id": "1",  
      "url": "http://{environment}/sdk/system/2/read/Department",  
      "body": {  
        "filter": {  
          "Syscode": {"eq": 260 }  
        }  
      },  
      {  
        "id": "2",
```

```
"url": "http://{environment}/sdk/system/2/read/Person",
"body": {
  "filter": {
    "Syscode": {"eq": 460 }
  }
}
]
```

## Response

The response will contain a response for each request.

```
{
  "responses": [
    {
      "id": "1",
      "status": "200",
      "body": {
        "records": {
          "records": [
            {
              "City": "Boxmeer",
              "Country": "Nowhere land",
              "Syscode": 260
            }
          ]
        }
      }
    }
  ]
}
```

```

},
{
  "id": "2",
  "status": "200",
  "body": {
    "records": {
      "records": [
        {
          "firstname": "Peter",
          "lastname": "Billings",
          "Syscode": 460
        }
      ]
    }
  }
}
}
}
}
}

```

## Download endpoint

The *Download*-endpoint makes it possible to download documents or images attached to a business object. The encrypted path and file name can be retrieved through a Read-request.

If the download endpoint is called the file will be automatically downloaded.

```

{
  "records": [
    {
      "Code": "304",
      "Email": "Alice.Wilson@planon.co.uk",
      "FirstName": "Alice",

```

```

      "LastName": "Wilson",

      "PhotoRef":
      "UGVyc29uLzMyL1Bob3RvUmVmL2h0dHA6Ly90b21jYXQtd2ViZGF2OjgwODEvd2ViZGF2L0ltYWdlcy9QZXJzb25uZWV067.jpg",

      "Syscode": 32,

    }

  ]

}

```

download request

```

http://{environment}/sdk/system/{version}/download/
UGVyc29uLzMyL1Bob3RvUmVmL2h0dHA6Ly90b21jYXQtd2ViZGF2OjgwODEvd2ViZGF2L0ltYWdlcy9QZXJzb25uZWV067.jpg

```

## Upload endpoint

Uploading a file in REST API is a two-step process.

First, a user uses the *Upload*-endpoint to get the file in an inbound box and a UUID is returned. Subsequently, this UUID is set to the appropriate field in an Update-request.

The REST API will retrieve the file from the inbound box and place it in the correct place and store the actual file path in the business object.

When sending a file to this endpoint, the *Content-Type* in the header should be set to **application/octet-stream**. When this request is valid, a UUID will be returned that can be used in an Update request (see example).

### Example

```

{
  "filter" : {
    "Code" : {"eq" : "304"}
  },
  "values" : {
    "PhotoRef": {
      "filename": "alice_wilson.jpg",

```

```
"uuid": "9df99275-2fcb-4541-9e3e-0c00b8022b81"
```

```
}
```

```
}
```

```
}
```

# Index

## A

- Assets
  - Attributes 49
- Attribute set
  - Adding values 51
  - Attach 50
  - Detaching 51
- Attribute values
  - Reading 53
  - Removing 53

## B

- Batch endpoint 73
- Body
  - Answers 43
  - Arguments 46
  - Confirmations 44
  - Filters 40
  - Values 42
  - Warnings 43

## C

- ChangeState endpoint 68

## D

- Definitions
  - Creating definitions 16
  - Multi-level 17
  - Single level 16
  - Using definitions 19
- Delete endpoint 65
- Download endpoint 75

## E

- Endpoint
  - Batch 73
  - ChangeState 35, 68
  - Create 27
  - Delete 33, 65
  - Download 75
  - Execute 66
  - Lookup 30
  - OpenAPI 69
  - Read 30, 62

- Update 31, 63
- Upload 76
- Endpoint patterns 62
- Endpoint
  - Lookup 67
- Endpoints
  - Examples 26
- Execute endpoint 66

## F

- File handling 48
  - Attach/Upload file 38
  - Remove file 48
  - Retrieve/Download file 36

## L

- Lookup endpoint 67

## O

- OpenAPI
  - Info 23
  - property 23
  - security 25, 25
  - servers 24
- OpenAPI document 23
  - Generate and import 19
- OpenAPI endpoint 69

## R

- Read endpoint 62
- ReferenceDate 59
- Request body
  - Detailed information 40
- Request header
  - reference date 59
- REST API
  - About... 8
  - Advanced features 48
  - Basics 12
  - BOs allowed 16
  - BOs and Fields 15
  - Definitions level 12
  - Field definitions level 13
  - Fields allowed 16
  - Introduction 7, 7
  - License usage level 14
  - Licentse 10

Methods 8  
Navigation panel 10  
Prerequisites 10  
Principles 7  
Security 11  
User access 11  
User interface 12

## **S**

Sample request  
  Creating 20

## **U**

Update endpoint 63  
Upload endpoint 76