Fabasoft

White Paper

Fabasoft Integration for OData

2026 November Release

Copyright © Fabasoft R&D GmbH, Linz, Austria, 2026. All rights reserved. All hardware and software names used are registered trade names and/or registered trademarks of the respective manufacturers.

No rights to our software or our professional services, or results of our professional services, or other protected rights can be based on the handing over and presentation of these documents.

Contents

1 Introduction 4

2 Installation of the Fabasoft OData Service 4

3 Basic Functionality 4

3.1 Creating an OData Service 4

3.2 Query Types 5

3.2.1 Found Objects 5

3.2.2 Referenced Objects 6

3.2.3 User Query 7

3.2.4 Examples Found Objects vs. Referenced Objects vs. Own Query 7

3.2.5 User Query Together With Filter Options 8

3.3 Filter Options 9

3.3.1 objchangedat 9

3.3.2 objaddress 10

3.3.3 Object_Source 11

3.4 Other OData Parameters 12

3.5 Creating an API Token 12

3.6 Licensing 13

4 Mapping of Fabasphere Objects to OData Entities 13

4.1 Mapping of Names 13

4.1.1 “Use Programming Names” Option 13

4.2 Attribute Value Generation 14

4.2.1 Basic Types 14

4.2.2 Lists 14

4.2.3 Aggregates 14

5 In-Depth Explanation: Execution of OData Requests 15

5.1 OData GET Request to an OData Service 15

5.2 OData GET Request to an Entity of an OData Service 15

5.2.1 Calculating the Cube 15

5.2.2 Getting the Attribute Values 15

5.3 OData Precalc (POST or GET) 16

5.3.1 GET Requests: Status Monitoring 16

5.3.2 POST Requests: Triggering Execution 16

5.3.3 Concurrency and Cache Behavior 16
6 Access the OData Service via Microsoft Power BI 16
7 Access the OData Service as a Developer 19
8 Performance 20
8.1 Performance of the Initial Call to an OData Service 20
8.2 Performance of Subsequent Calls to OData Service 20
8.3 Performance Optimization “Getting Only X Rows” 21
8.4 Precalculation 21
8.4.1 Precalc and Data Filtering 22
9 Restrictions 22
9.1 Data Types 22
9.2 Categories and Entities 22
9.3 User Forms 23

1 Introduction

With the Fabasoft Integration for OData (https://www.odata.org), structured data can be accessed.

This document describes the setup of an OData service as well as the access via Microsoft Power BI. For detailed instructions on how to generate sophisticated reports with charts, lists, relations, etc., please check out the official Microsoft Power BI documentation.

2 Installation of the Fabasoft OData Service

For the time being, the Fabasoft OData service functionality must be enabled by Fabasphere Support (cloudsupport@fabasoft.com).

3 Basic Functionality

An OData service can be created in apps that have reporting enabled (e.g. Scrum). The OData service contains the definition of the object classes and properties that are exposed via the OData API.

Depending on where you create the OData service, various source objects can be used. For example, in order to gain access to your contracts, create an OData service inside your Contract Manager app.

Having created an OData service, you receive a URL which can then be used to access the data via a client like Microsoft Power BI. The OData protocol is a standardized protocol which is understood by any client - even a web browser.

Aside from the OData URL, you require the API token for authentication.

3.1 Creating an OData Service

Once reporting has been enabled in the app of your choice, you need to create the OData service in the context where your data is stored so that it can be queried.

To create an OData service, proceed as follows:

1. Navigate into the desired app configuration or app room, then into the "Reports and OData Services" area.
2. Click the "Create OData Service" action.
3. Define the following settings and click "Next".
   - Name
     The name of the OData service.
   - Language
     Multilingual strings are provided in the defined language in the data source.
   - Use Programming Name
     If enabled, object classes and properties will not be translated and the programming names are used instead.

- Cache Duration
  Defines how long the data is cached. Defining longer time intervals will increase performance during subsequent calls.

4. You can choose whether to use a source object to define the OData service model or to define the model manually. Both options are described below.
5. Option "Use Source Object"
   5.1. Enable the *Use Source Object* option to automatically determine object classes and properties using an initial source object. The source object and its contained objects are used to define the OData service model. The source object can be one of the following: `Group`, `AppRoom`, `AppConfigurationRoom`.
   5.2. Select a source object to determine the object classes and their properties for the OData service model and click "Next".
   5.3. The object classes and properties are preconfigured. You can customize the model to suit your needs.
6. Option "Manual Selection"
   6.1. Click "Next" without enabling the *Use Source Object* option to manually define the model.
   6.2. Choose the object class of which you want to add properties to the model for the OData service and click "Next".
   6.3. Select the desired properties and click "Add Selected".
   Note: You need to consider that adding a great number of properties will require additional resources to load. It is recommended to add the minimum required number of classes and properties.
   6.4. Click "Next".
7. If needed, you can add further properties of other object classes by clicking "Add Entry".
8. Click "Next".
9. Copy the provided link to the OData service and click "Next".
   Note: You will be able to copy this link later as well by clicking on the generated OData service.

3.2 Query Types

For each entity you can select between found objects, referenced objects and user query. For "Found Objects" a query is defined where these objects are searched for. Referenced Objects are just objects referenced by another attribute.

There the following restrictions are applied: for the object classes `#Object`, `#User` and `#Group` only "Referenced Objects" is possible. In general, also the permission `#AccTypeSearch` must be given to be able to select "Found Objects".

3.2.1 Found Objects

"Found Objects" means that the objects of the entity are determined by an internal search query. For each entity with this option, the OData service runs a query that finds all objects of

the entity's object class (and selected category) within the service's restriction context (app room, app configuration or organization, depending on the restriction object defined when the OData service was created).

The following happens when the data of the service is calculated:

- **Execute search query:** The search query is executed (a standard internal query generated from the object class, the category and the service's restriction context or a user query of the entity, see chapter 3.2.3 "User Query"). All objects the query matches are part of the result and they are the "`found`" objects.
- **Discover Referenced objects:** The attributes of every "`found`" object that are defined in the OData service are then examined. Every object that is referenced by one of these attributes and that belongs to the same entity (same object class and category) is added to the result as well.
- **Decide to continue recursion with Referenced objects:** Whether the process continues with the newly added objects depends on the OData service setting "Advanced Configuration: No Recursive Following of Properties" (avoid recursion):
  - **If avoid recursion is not set:** The attributes of the newly added objects are examined in turn. Objects they reference that belong to the same entity are added as well and so on, until no new objects of the entity are found. In this way a complete chain of mutually linked objects is returned, starting from the objects the search found.
  - **If avoid recursion is set:** The attributes of the newly added (referenced) objects are not examined. Only the objects found by the search + the objects directly referenced by them are returned. This can dramatically improve the performance depending on reference structures present in the organization.

Note: Every object is marked with the source "search" and/or "referenced" depending on how it was obtained. This source can be used to filter the result, see chapter 3.3.3 "Object_Source".

3.2.2 Referenced Objects

"Referenced Objects" means that no search is executed for the entity. Instead, the objects of the "Referenced Objects" entity are collected from the references of the objects that were obtained by executing "Found Objects" or "User Query" entities within the OData service.

Concretely, an object X appears in an entity E with "Referenced Objects" of an OData service if 1) X has the object class (and category) of E, **and** 2) at least one object of another entity Y (obtained by executing that entity Y belonging to "Found Objects" or "User Query") holds a reference to X in one of the attributes defined in the OData service for the entity Y.

Example: An OData service is defined with entities "Document" with "Found Objects", "Images" with "User Query", and "User" with "Referenced Objects". The document search finds the documents, and the attribute "Created by" is part of the "Document" entity. Similarly, user query search finds the images, and again the attribute "Created by" is part of the "Images" entity. Every user referenced via that attribute "Created by" from objects found in both "Document" and "Images" entity then appears in the "User" entity even if the users were not found by any search.

3.2.2.1 "Object" entity

The base class #Object is always part of an OData service and is fixed to be a "Referenced Objects" entity type. It collects the referenced objects whose class is not itself an entity of the OData service. That means: whenever a "Found Objects" or "User Query" entity is executed and the resultant object holds a reference to an object of a class that have not been defined as an entity, the referenced object is still added to the set and can be found in the "Object" entity. This way no reference data is lost, but the details of such objects are limited to the attributes that have been added to the "Object" entity (e.g. `objaddress`, `objname`, `objclass`).

Example: An OData service is defined with entities "Document" with "Found Objects" and "User" with "Referenced Objects". "Object" entity will be automatically added to the OData service. The document search finds the documents, and the attribute "Team room" is part of the "Document" entity. Every team room referenced via that attribute "Team room" then appears in the "Object" entity since the "Team room" was not defined as an entity in the OData service.

3.2.3 User Query

The user query enables you to create own queries which narrows down the found objects.

Example:

```
SELECT * FROM COMPONENT@100.200:EntityType WHERE .FSCTEAMROOM@1.1001:objteamroom IN
(COO.100.200.300.xxxx,COO.100.200.300.xxxx) AND .objexternalkey LIKE 'x1%'
```

This query would narrow down the objects to a Teamroom and looks through the attribute `objexternalkey`. The referenced objects will work the same, just with the objects which were filtered with the user query.

3.2.4 Examples Found Objects vs. Referenced Objects vs. Own Query

These examples should demonstrate the decision between “Found Objects” and “Referenced Objects”. In general, if you are unsure, “Found Objects” can be a good start as you will get data there. Only if you want to restrict the found objects, switch to the option “Referenced Objects”.

Example 1: Books used by Machines

Imagine there are 10 000 registered books in my organization. Some books (like manuals, about 100) are for maintaining my machines.

I want to find with OData my machines and the books used for their maintenance.

So, I define an OData Service with restriction to my whole organization with two entities:

1. Machine with option “Found Objects” because I want that all my machines are found via query.
   1.1. I add there the attribute “Manual” which refers to the book used for maintenance.
2. Book with option “Referenced Objects” because I only want to get books which are referenced by the “Manual” attribute from my machines.

The outcome is that the “Machine” entity contains all my machines and the “Book” contains all manuals referenced in my machines (about 100).

Example 2: Full list of Books and Machines

In comparison to the first example, I want to do statistics on my inventory. For that I need all machines and all books I have in the Fabasphere.

So, I define both entities with the option "Found Objects". The result will be all my machines and all my books (about 10 000). Further filters I could apply with my Business Intelligence Tool like Microsoft Power BI.

Example 3: Machines which have been built before 2021

Imagine we want to monitor only machines, which have been built before 2021, "Found Objects" and "Referenced Objects" does not help us because we are always loading all machines.

For such a filtering of the input data, user queries have been introduced as described in chapter 3.2.3 "User Query".

```
SELECT * FROM COMPONENT@100.200:Machine WHERE .FSCTEAMROOM@1.1001:objteamroom IN
(COO.100.200.300.xxxx,COO.100.200.300.xxxx) AND .builtatyear < 2021
```

3.2.5 User Query Together With Filter Options

If no user query is provided, a query will be created automatically. The filter options like

```
/Story?$filter=objchangedat gteq 2024-06-28T11:55:46
```

are then added automatically to the query.

But what if you provide a user query? It cannot be determined automatically, where in your user query to add the filter criteria.

Example:

You have a user query

```
SELECT * FROM FSCSCRUM@1.1001:Story WHERE .objname LIKE "%Storage%" AND .objname
LIKE "%Service%"
```

If you want to request this OData-Service with a top filter or a query filter, the Fabasoft OData does not know how to modify your query to include these both criteria.

That is why you have to add the placeholders

```
[TOPFILTER]
```

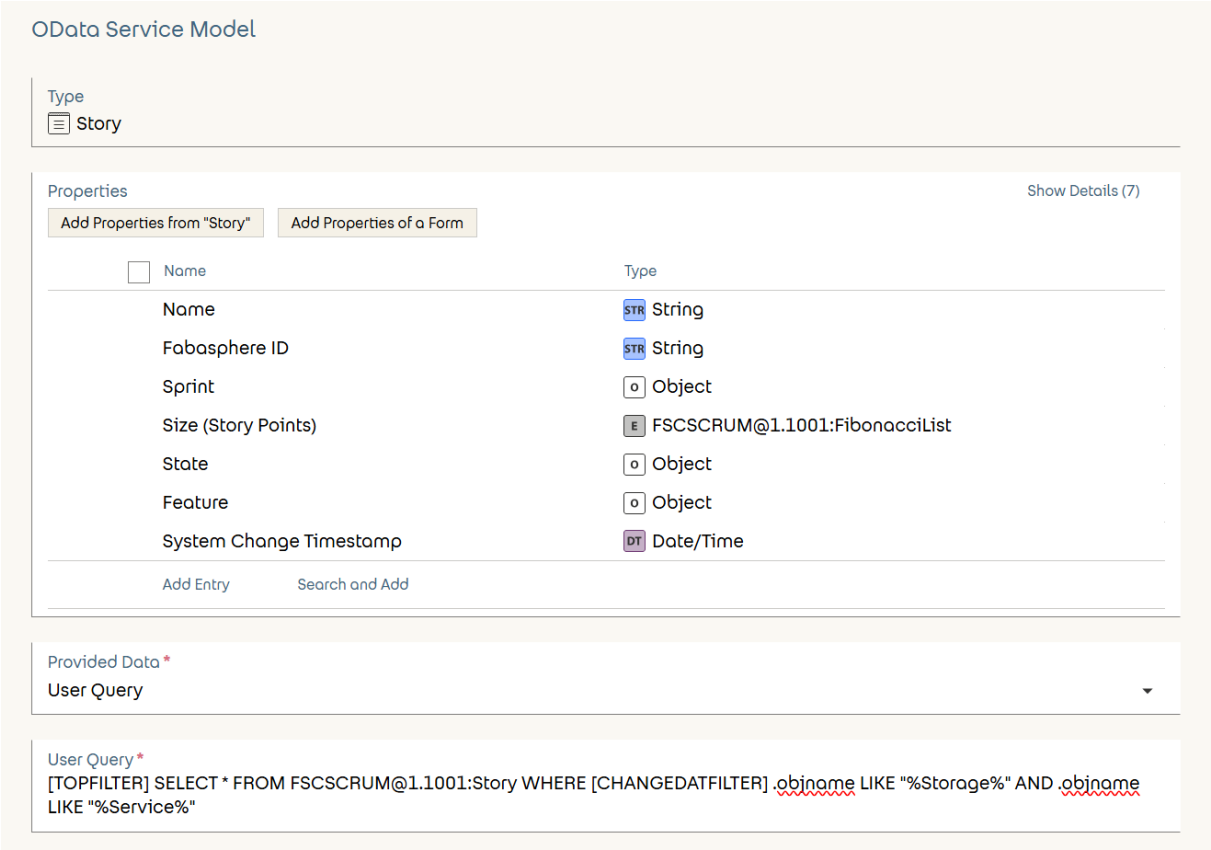
and

```
[CHANGEDATFILTER]
```

If we want to enable the `$top=` option as well as the `$filter=` option, we have to add both placeholders to our defined user query:

```
[TOPFILTER] SELECT * FROM FSCSCRUM@1.1001:Story WHERE [CHANGEDATFILTER] .objname
LIKE "%Storage%" AND .objname LIKE "%Service%"
```

In the OData dialog it can look like that:



3.3 Filter Options

Fabasoft OData does support a subset of possible filters with the goal to optimize the performance. The supported filters are `$top` (see chapter 8.3 'Performance Optimization "Getting Only X Rows"') and `$filter` on the attributes `objchangedat` (chapter 3.3.1 "objchangedat"), and its translated names or the attribute `objaddress` (chapter 3.3.2 "objaddress") and its translated names. `$filter` also supports `Object_Source` (chapter 3.3.3 "Object_Source") filter. These filter clauses `objchangedat` and `objaddress` can only be combined with `Object_Source` filter using the "`and`" operator. It is not supported to combine both of them (`objchangedat` and `objaddress`) in one single filter statement.

3.3.1 objchangedat

To filter by `objchangedat` the attribute `objchangedat` (System_Change_Timestamp) has to be included in the attributes of an entity. Furthermore, filtering is only possible with the `gteq` (or `ge`) , `gt`, `lt` and `lteq` operators.

The syntax is:

```
/<entity>?$filter=<objchangedat or translated name> gt|ge|gteq|lt|lteq <date and time with or without timezone>
```

Note: `objchangedat` would be the right name if your OData Service has "Use Programming name". If not, it depends on the language of the OData Service. Means for an English OData Service the filter would look like `filter=System_Change_Timestamp…` and in German it would look like `filter=System_Änderungszeitpunkt…`

An example query could look like this:

```
/Story?$filter=objchangedat gt 2024-06-28T11:55:46
/Story?$filter=objchangedat gteq 2024-06-28T11:55:46
/Story?$filter=objchangedat lt 2024-06-28T11:55:46
/Story?$filter=objchangedat lteq 2024-06-28T11:55:46
```

also possible with a time zone (be sure you encode the "+" with a "%2B"):

```
/Story?$filter=objchangedat gteq 2024-06-28T11:55:46%2B02:00
```

The `$filter` and `$top` filter can be combined by using `&`, for example:

```
/Story?$filter=objchangedat gteq 2024-05-28T11:55:46&$top=15
```

3.3.1.1 Combining objchangedat Filters

When using `objchangedat` in a `$filter` statement, certain operator combinations in a single filter statement are not supported:

- `gt <date>` cannot be combined with `ge` or `gteq` using `and` or `or`
- `lt <date>` cannot be combined with `lteq` using `and` or `or`

Examples of unsupported combinations:

```
/Story?$filter=objchangedat gt 2024-06-01T00:00:00 and objchangedat gteq 2024-06-01T00:00:00
/Story?$filter=objchangedat lt 2024-06-30T23:59:59 or objchangedat lteq 2024-06-30T23:59:59
```

All **other** operator combinations are supported.

For example:

```
/Story?$filter=objchangedat gt 2024-06-01T00:00:00 and objchangedat lt 2024-06-30T23:59:59
/Story?$filter=objchangedat ge 2024-06-01T00:00:00 or objchangedat lt 2024-06-30T23:59:59
```

3.3.2 objaddress

To filter by `objaddress` the attribute `objaddress` (Fabasphere ID) has to be included in the attributes of an entity. Furthermore, filtering is only possible with `in` operator.

The syntax is:

```
/<entity>?$filter=<objaddress or Fabasphere ID> in ('<COO address>')
```

Notes:

The `in ('')` filter supports both a single Fabasphere Object Address or a list of addresses. You must use single quotes ' (not double quotes ") around each value.

An example query could look like this:

```
/Story?$filter= objaddress in ('COO.1.30000.1.14755', 'COO.1.30000.1.14756')
```

`objaddress` would be the right name if your OData Service has “Use Programming name”. If not, unlike other translated attributes `objaddress` remains across all languages.

An example query could look like this:

```
/Story?$filter= objaddress in ('COO.1.30000.1.14755')
```

3.3.3 Object_Source

When the OData service is built, every returned object has a "source": 1) it is either found directly by the search query of the entity (search query generated internally for "Found Objects", see chapter 3.2.1 “Found Objects” and with provided search query in “User Query”, see chapter 3.2.3 “User Query”), or 2) it was pulled in because another returned object, references it within its attributes. The `Object_Source` filter allows restricting the returned objects to one of these sources.

The filter attribute is `Object_Source`. Unlike other filter attributes, it is not a mapped property of the entity and does not need to be added to the entity's attribute list. It is available for every entity. Also, it has no translated names and should be used as is.

Supported values:

- `SOURCE_SEARCH`
  only objects found by the search query
- `SOURCE_REFERENCED`
  only objects that are referenced in one of the attributes of the returned object.

Objects that are both searched and referenced are returned for both values. There is no value for "both sources". If the `Object_Source` filter is not provided, all objects are returned (the default option).

The syntax is:

```
/<entity>?$filter=Object_Source eq SOURCE_SEARCH
/<entity>?$filter=Object_Source eq SOURCE_REFERENCED
```

Notes:

- Only the operator "`eq`" is supported.
- The value names are case insensitive (e.g. `'source_search'` works as well).
- `Object_Source` can be combined with the other filters (`objchangedat`, `objaddress`) using the "`and`" operator only. Combining it with "`or`" is rejected. `Object_Source` should be at the end of this filter combination i.e. it should either in front of the other filters or at the end of these filters.
  Examples:
  ```
  /Story?$filter= Object_Source eq SOURCE_SEARCH and objchangedat gt 2024-06-01T00:00:00 and objchangedat lt 2024-06-30T23:59:59
  /Story?$filter= objchangedat gt 2024-06-01T00:00:00 and objchangedat lt 2024-06-30T23:59:59 and Object_Source eq SOURCE_SEARCH
  /Story?$filter= objaddress in ('COO.1.30000.1.14755', 'COO.1.30000.1.14756') and Object_Source eq 'SOURCE_REFERENCED'
  ```

- The filter also works on cached/precalculated data (see chapter 8.4.1 “Precalc and Data Filtering”). When the data is served from the cache, the objects of the entity already carry their source (search or referenced), because this information is stored together with the cached data. Applying `Object_Source` filter on such data therefore adds no calculation overhead and thus separating "`SEARCH`" and "`REFERENCED`" objects is just a selection from the already loaded result set.
- Without precalculation or cache, the same separation is slightly more costly: to return only the searched objects, the service would have to run the search query once, and to return only the referenced objects it would have to run it a second time (collecting the references without the searched objects). Thus, the same search would effectively be performed twice for the two partitions. The reference traversal is only performed when `'SOURCE_REFERENCED'` is requested or the when the filter `Object_Source` is not applied. With cached data, both partitions can be served from the same single calculation.

Behavior:

- With `SOURCE_SEARCH`, an object is returned only if the search query of the entity matched it directly.
- With `SOURCE_REFERENCED`, an object is returned only if at least one returned object references it. An object that is both searched and referenced is returned by both filters.
- The union of both filter results is exactly the set of objects a normal (unfiltered) query returns; the intersection is the set of searched objects that are also referenced.
- Objects whose source cannot be determined (e.g. from a cache written before this feature existed) are not returned when `Object_Source` filter is used.
- `Object_Source` filter is redundant in case of "Referenced Objects" entity type. Since all objects of "Referenced Objects" entity type are found via references only.

3.4 Other OData Parameters

OData knows a lot of other parameters. The Fabasoft OData supports only a subset of them:

- `$select`
  Supported. You can e.g. define the attributes you want to get back like `$select=name,id`
- `$top`
  Supported (see chapter 8.3 ‘Performance Optimization “Getting Only X Rows”’).
- `$filter`
  Supported. This is not affecting the performance as still all attributes are calculated. It only reduces the amount of data coming back.
- Other parameters like `$skip`, `$expand`, `$orderby` and so on are **not** supported. They will be ignored.

3.5 Creating an API Token

You need an API token to access your OData service. Create an API token by opening the account menu (your user name) and clicking “Settings” > “Access for Applications”. Create an API token valid for “Open Data Protocol (OData)”.

3.6 Licensing

OData licenses are available in 10M and 100M versions.

OData volume licenses are checked when

- the table overview for an OData service is retrieved.
- actual rows for a certain entity are retrieved.

The overview does not lower the number of available objects but the overview is not shown if there is no amount of any OData volume left.

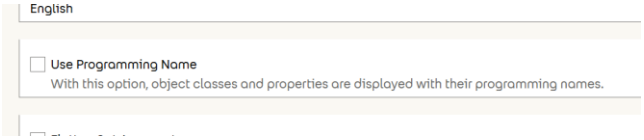
When retrieving rows for a certain entity the OData volume is lowered by the number of rows retrieved. That means you are charged exactly by the number of rows retrieved no matter which filters are applied and how many data rows there are in original.

4 Mapping of Fabasphere Objects to OData Entities

The input data are always objects in the Fabasphere. The OData service has the following strategy to map objects and attributes of objects to the OData entities and OData properties.

4.1 Mapping of Names

Depending if you have checked if you have checked the “Use Programming Names” option when creating an OData service, the names are generated differently.



4.1.1 “Use Programming Names” Option

When “Use Programming Names” is checked, the internal reference names of Fabasphere objects and attributes are used as seen in this screenshot of the JSON output of the OData service:

If the option is not checked, the display names in the language select in the OData service is used.

The following rules apply:

- Special characters including whitespaces are replaced by an underscore.
- Leading and trailing whitespaces are not trimmed, the normal replacing logic applies.

```
{
    "@odata.context": "https://at.cloud.fabasoft.com/odata/COO.6505.100.13.570666/$metadata#Story",
    "value": [
        {
            "Fabasoft_Cloud_ID": "COO.6505.6000.3.8906843",
            "Current_Stage": null,
            "Subject": null,
            "Lead": null,
            "Release": null,
            "Stakeholders": [],
            "State": "COO.1.1001.1.219833",
            "Sprint": null,
            "Scrum_Project": "COO.6505.100.9.8766663"
        },
```


4.2 Attribute Value Generation

Values of attributes in the Fabasphere are mapped into a value in OData. Here the following rules apply:

4.2.1 Basic Types

- Basic data types are mapped directly. Integer types get an integer type in OData. String types get a string type in OData.
- The only exception here is the basic type "Currency" as this is also an aggregate where special rules apply (see next point).

4.2.2 Lists

- Lists are mapped directly into OData/JSON lists.

4.2.3 Aggregates

- Aggregates are flattened out. That happens by concatenating the aggregate name with the attribute name with an underscore as separator.
  So, if you have an aggregate named `Lock Information` and there inside the attributes `Is Locked`, `Locked By` and `Locked Since` it will look like that:


```
{
  "@odata.context": "https://at.cloud.fabasoft.com/odata/COO.6505.1.8.1
  "value": [
    {
      "Lock_Information_Is_Locked": "true",
      "Lock_Information_Locked_By": "COO.6505.100.10.8151285",
      "Lock_Information_Locked_Since": "2023-07-04T14:08:40+01:00"
    }
  ]
}
```


  Important is, that these 3 properties are only here, if the `Lock Information` aggregate is set at all. So, it if is null, these 3 properties won't be in the result set.

- Currencies are aggregates as well. They are flattened out with the two options `currvalue` and `currsymbol` so if you have a currency data type on an attribute `Price` it could look like that:


```
{
  "@odata.context": "https://at.cloud.fabasoft
  "value": [
    {
      "Price_currvalue": "94.06000000000000",
      "Price_currsymbol": "Euro"
    }
  ]
}
```

5 In-Depth Explanation: Execution of OData Requests

There are three main types of requests possible:

1. OData GET request to an OData Service
2. OData GET request to an entity of an OData Service
3. OData precalc (POST or GET)

5.1 OData GET Request to an OData Service

When OData GET requests are done, the OData Service in the Fabasphere is loaded and the needed information about the OData schema are calculated. The result is a valid OData overview.

5.2 OData GET Request to an Entity of an OData Service

On such a request, actually data is loading from the Fabasphere and given back in OData style.

The whole process can be split up in these parts:

1. Generating the schema.
2. Calculating the cube.
3. Getting the attribute values.
4. Building the OData response.

Especially the second and the third step can be performance critical, so these are described in more detail.

5.2.1 Calculating the Cube

Given an entity defined in the OData Service as "Found Objects", two queries to the Fabasphere are done to fetch the data:

First a query is done which looks up all Teamrooms in the given context (room, configuration or organization). This list of Teamrooms is then used in the second query, which looks up objects of the entity type which are in the given list of Teamrooms.

Important: Please be careful here about limits: if your selected context (room, configuration or organization) has a lot of Teamrooms (10 000+), this can cause the second query which is an IN-Query, to not run performant or not run at all because of too many IN-values. Please consider here defining your own query for this entity!

5.2.2 Getting the Attribute Values

In parallel running threads/processes, the wanted attributes are loaded. On performance issues please check out the performance chapter.

5.3 OData Precalc (POST or GET)

The OData precalc endpoints manage the asynchronous pre-calculation of OData service definitions. They allow users to trigger data processing in the background and monitor execution status.

5.3.1 GET Requests: Status Monitoring

GET requests retrieve live status information for pre-calculation tasks. The response details vary based on the endpoint structure:

- Service-Level Status
  Sending a GET request to `/odata/precalc/{cooid}` returns the execution status of all current jobs calculating data for that specific OData service definition.
- Job-Level Status
  Sending a GET request to `/odata/precalc/{cooid}/{id}` returns detailed metrics and status for that specific background job instance.

An empty response indicates that all associated job information related to the service definition has been automatically removed after the system configured time to live of the status information.

5.3.2 POST Requests: Triggering Execution

Sending a POST request to `/odata/precalc/{cooid}` initiates the pre-calculation workflow for the specified OData service definition. The endpoint immediately returns a unique job ID (`{id}`), while the actual calculation runs asynchronously in the background.

- Processing
  The system processes the service definition and calculates all associated entities.
- Storage
  The calculated data is saved to the Teamroom defined in the OData service configuration.

5.3.3 Concurrency and Cache Behavior

If multiple POST requests are sent for the same OData service definition (`{cooid}`), the system generates multiple distinct background jobs.

- Execution
  These jobs process the pre-calculations concurrently across different threads.
- Storage Impact
  The parallel jobs overwrite the same shared cache destination in the cloud.

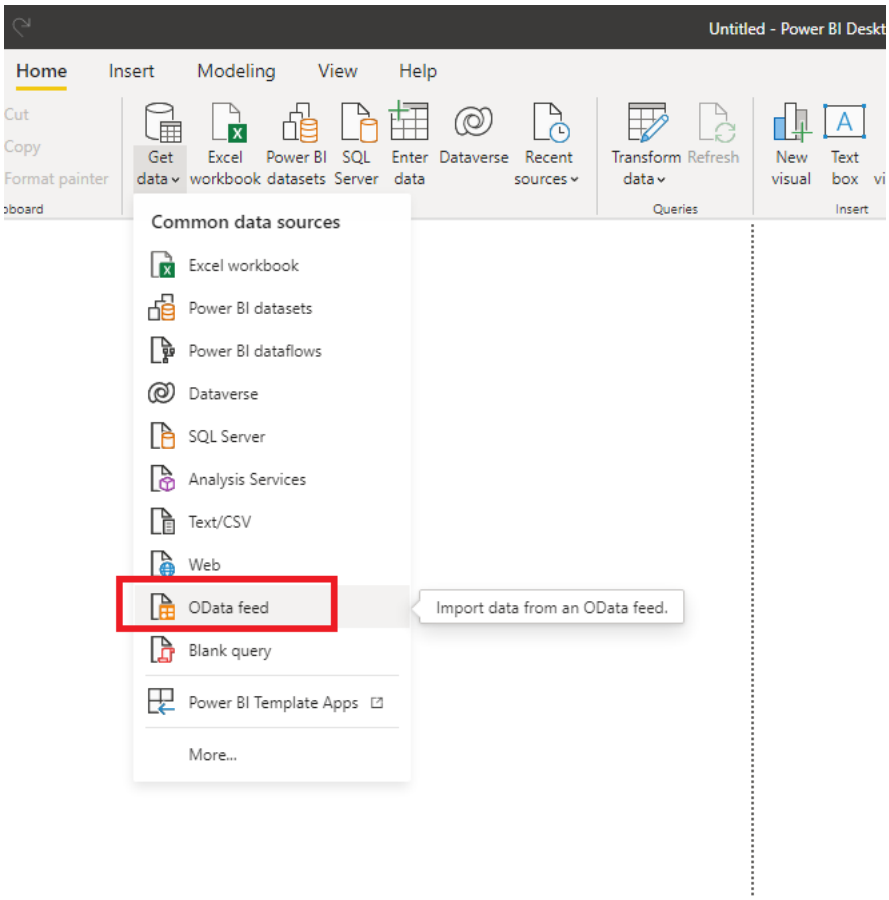
6 Access the OData Service via Microsoft Power BI

Using Microsoft Power BI to access the data provided by the OData service is quick and easy. Microsoft Power BI is a great tool for creating reports and statistics.

For more details on how to work with Microsoft Power BI please refer to:
https://powerbi.microsoft.com/en-us/support/

To add the OData service to Microsoft Power BI, proceed as follows:

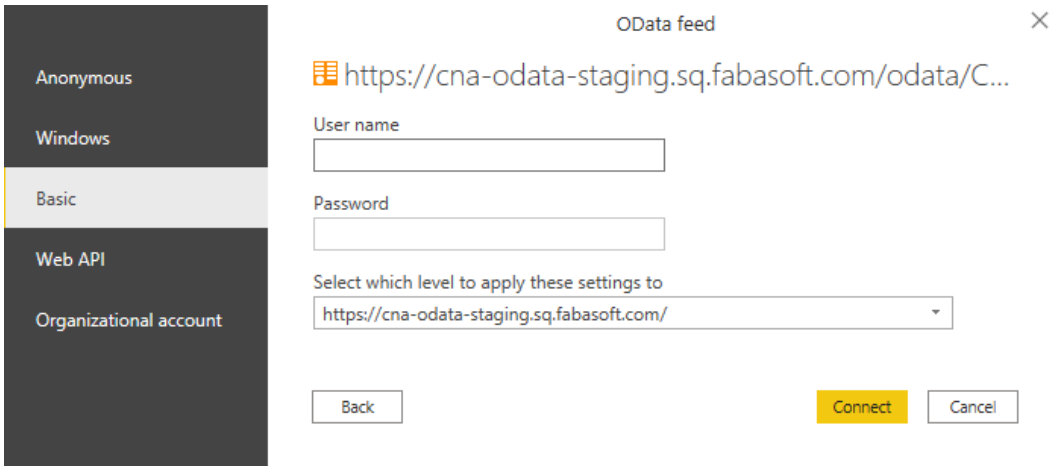
1. On the "Home" tab, click the "Get data" button, and then click "OData feed" as data source.



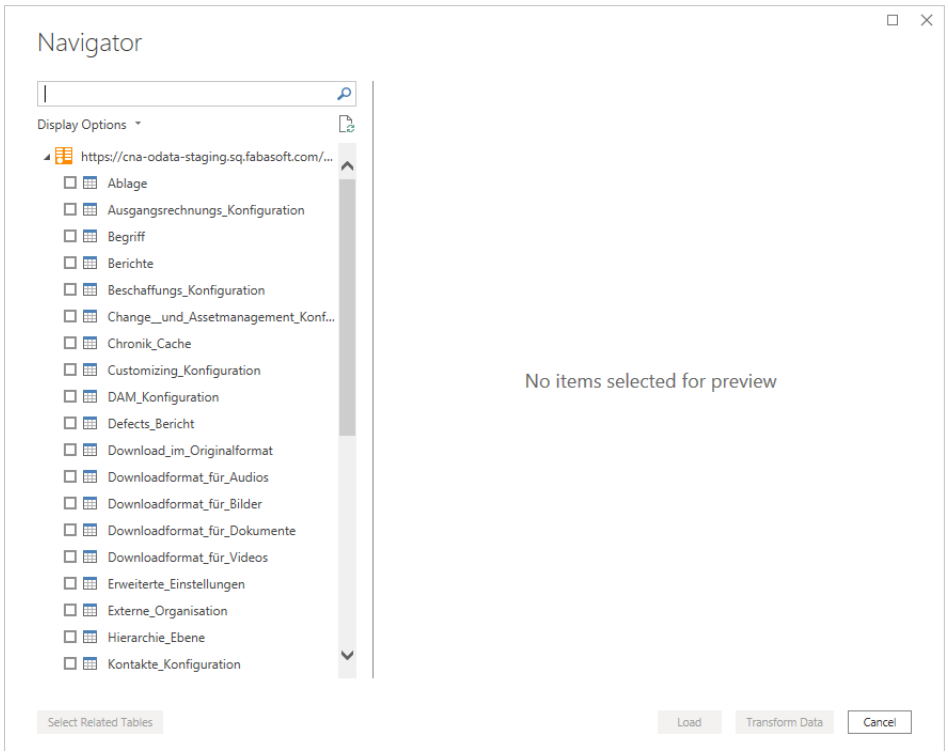
2. Enter the URL of the OData service (see chapter 3.1 "Creating an OData Service") in the *URL* field and click "OK".
   Note: The "Basic" connection option is sufficient.

   Since OData usage requires authentication you need to select the "Basic" authentication option. Enter your e-mail address for the Fabasphere log-in as the user name and the generated API token as the password (see chapter 3.5 "Creating an API Token").

3. Click “Connect”.



4. After successful authentication, you can select from several display options in the “Navigator”. Select your desired options and click the “Load” button.

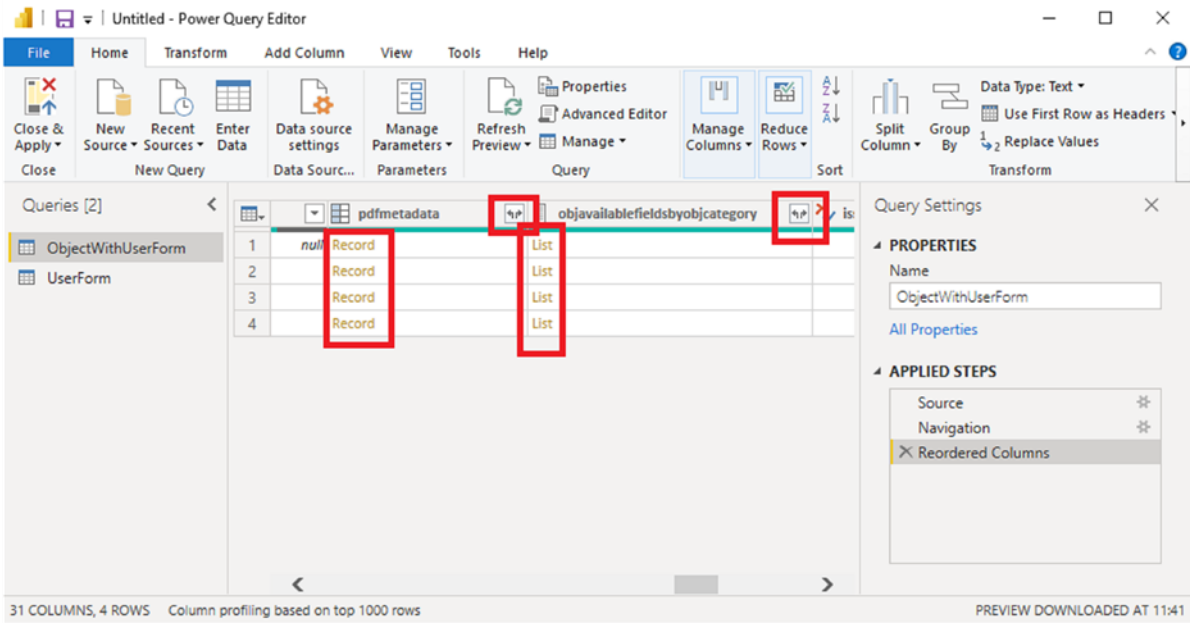


Lists

When you create an OData service, it is very likely that some of the required properties have lists as data types.

Lists are displayed as “List”. In the “Transform Data” dialog you can see these lists.

In order to make them useable, click on the “Expand” tool at the end of the column (see red box on the column in the screenshot) to choose an expanding option. Such an option is e.g. creating a new table; however, this requires additional work as relationships between the tables need to be set.



For lists, the approach to make them useable depends on the expected number of entries. If you can predict having only a couple of entries, expanding the list is an option. In this case, the other values are duplicated. If there are many entries, creating a new table based on the list may be a better solution.

7 Access the OData Service as a Developer

The OData protocol is a standardized HTTP-based protocol to retrieve data. Please check https://www.odata.org/ for details regarding the standard, programming examples and more.

When accessing an OData service URL via a web browser, you can interactively see the results of OData queries as JSON documents or XML documents.

Accessing data via an OData service is quite simple. Basically, all you need is to create an HTTP-based request with basic authentication turned on. The user name is the e-mail address and the password is the generated API token.

The response is always a JSON document. In many programming languages, there is already a good toolset to parse and interpret JSON.

Note: If you have to process the metadata as well, simply follow the URLs stored in `@odata.context` which you can see in the screenshot below.

8 Performance

The Fabasoft Integration for OData processes data in of two steps:

- During the first step (“Cube creation”), all objects as defined in the data source are queried and loaded into the OData service. The scope of objects considered is defined by the context of the data source (e.g. Teamroom). If the objects of the query result contain references to instances of classes contained in the data source, these instances are added to the object set. No queries are executed for the object classes “Object”, “User” and “Group”. Instead, the references to these instances are collected in the corresponding tables.
- During the second step (“Table creation”), the values per table are loaded and mapped to the structure as required by the OData query processor. This data is created per table on demand. However, all of the table rows are created independent of the amount of table rows queried.

Keep in mind that any caching within the OData services is performed per user. Thus, if multiple users access an OData service, they all work with their dedicated OData cache.

8.1 Performance of the Initial Call to an OData Service

In order to reduce the processing overhead of the initial call to an OData service you can

- reduce the number of entities or tables, and/or
- reduce the number of properties inside entities or tables with high row count.
  Try removing unused properties to reduce the overall processing time for each entity. This is most likely the fastest way to resolve such performance issues.

8.2 Performance of Subsequent Calls to OData Service

Should the OData service lack performance with regards to subsequent calls, you may increase the value in the *Cache Duration* field. In this case, the data is stored in the cache for a longer period, hence subsequent calls may reuse the already existing data table.

8.3 Performance Optimization "Getting Only X Rows"

In OData request syntax there is a filter targeting the number of rows fetched. It is the "top" command and can be used as followed:

```
odata/COO.1.506.4.2141/Scrum_Story?$top=500
```

This top command is also used in some business intelligence tools like Microsoft Power BI when showing a preview of the data.

Because it is commonly used this command has been optimized. That means internally only that amount of entities are fetched and processed. For the user it is transparent – it just does what it should do, just a little faster than other commands.

8.4 Precalculation

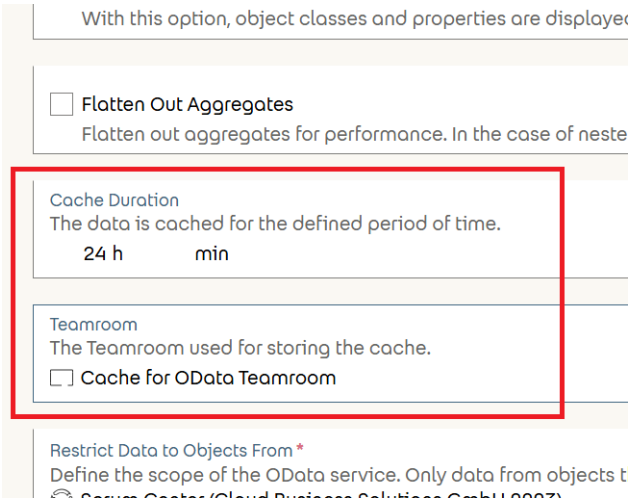
If reducing the number of attributes as well as splitting up to less entities per OData service does not help to get the result within the given time window (30 minutes), you can use precalculation.

Precalculation is triggered similar like calling an OData service, just with `precalc` in the URL and a HTTP POST instead of HTTP GET.

The precalculation is performed asynchronously on the server side. That means, when you trigger the precalculation for an OData service, you will quickly get back a 200 OK. In the background it will precalculate the OData service by writing a cache into a Teamroom in the Fabasphere.

To enable the precalculation, two steps are needed in the OData service:

- Ensure a valid cache duration is entered. 24 hours is our suggestion. That means, you could trigger the precalculation at night and then on the whole day these cached entries are taken to quickly deliver OData results.
- A Teamroom has to be entered below where the cached data is stored. Inside this Teamroom, another Teamroom will be created especially for you.
  Please be careful with giving additional permissions on these inner Teamroom to others as it may be that others can see data they normally have no access to.



Finally, you have to call the precalculation. If the normal URL for your OData service is

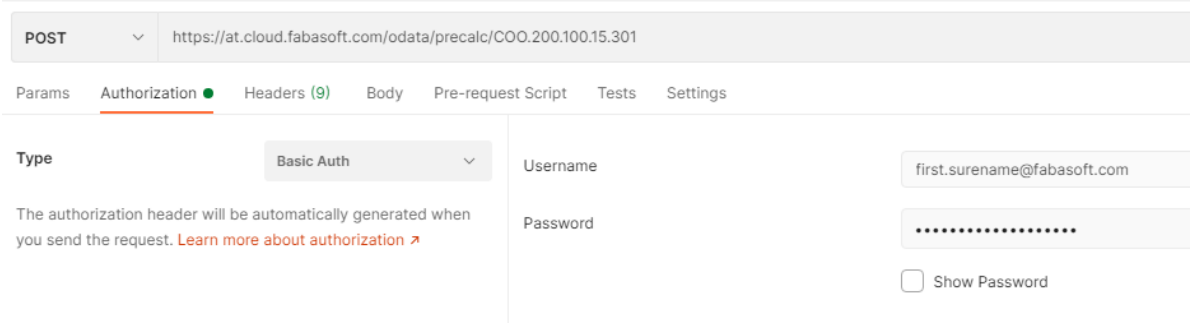
`GET https://at.cloud.fabasoft.com/odata/precalc/COO.200.100.15.301`

you have to add `/precalc` after `/odata` and use the HTTP `POST` method instead of HTTP `GET`.

This could be your final URL:

`https://at.cloud.fabasoft.com/odata/precalc/COO.200.100.15.301`

You can use e.g. `curl` or Postman to trigger that request. The authentication is the same as on a regular OData request – basic authentication with your username and the generated API token.



Depending on your OData service and your data, it can take hours to be finished. Triggering it at night time makes sense.

The actual GET OData request should then be way quicker compared to without preloading. Depending on the amount of returned data entries, it can then take seconds to minutes to be finished.

8.4.1 Precalc and Data Filtering

When you enable data retrieval from cache by activating `precalc` (as before), you can still apply data filtering as usual. Filter statements can be applied in the same way as described in chapter 3.3 "Filter Options". This includes the `Object_Source` filter (chapter 3.3.3 "Object_Source"): objects cached by a precalculation carry their source (search vs. referenced), so a later request with `$filter=Object_Source eq 'SOURCE_SEARCH'` and a request with `$filter=Object_Source eq 'SOURCE_REFERENCED'` can both be served from the same precalculated data. Without cache, each of the two partitions would require the search to run twice and the reference traversal to be run separately. Note that objects written to the cache before this feature was available have no stored source and are therefore not returned by an `Object_Source` filter.

9 Restrictions

The following restrictions apply.

9.1 Data Types

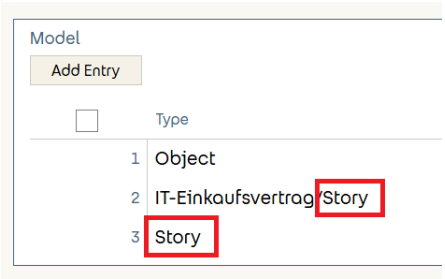
There is no support for content or dictionary properties.

9.2 Categories and Entities

Each entity can only be used once in an OData service.

In the following working example, you can see as entity a story which only appears if an "IT-Einkaufsvertrag" is chosen as category.

But this story **cannot** appear twice in the model:



9.3 User Forms

Only the following use cases with user forms are supported:

- Adding properties of an existing user form.
- User form data types: primitive data types, lists of strings.
- Adding properties of more than one user form.