# ***Using Pagination, Filter and Sorting query params in Horizon Server REST API***

Following up with Getting Started Guide, this guide serves the purpose on how to use Pagination, Filter and Sorting query params with Horizon Server REST APIs.

Refer to Swagger API Reference to know more about each API and Getting Started Guide on how to invoke Horizon server REST API.

Horizon Server REST API are hosted from the installed connection server instance and the Base URL Path for consuming this API is

https://<IP of connection server>/rest

***Base-Path : https://<IP or FQDN of connection server>/rest***

## ***Pagination***

### ***Overview***

Pagination parameters in REST API are enabled on selected List APIs, which is indicated in their respective swagger API reference.

Following key points needs to be considered for using Pagination –

* Any Pagination enabled List API will be capable of receiving two **optional** query params to support pagination - **page** and **size**.
* If pagination is intended, then BOTH page and size are mandatory parameters.
* If either page or size is present, the resulting list will not be paginated.
* No assumptions will be made for **page** defaulting to 1 if only **size** is present.
* **page** and **size**has to be positive (greater than 0) numeric literals only (Example /rest/<List API>?page=2&size=10).
* There is a limit set on maximum value of **size** which is **1000**.
* An optional response header '**HAS\_MORE\_RECORDS**' will be made available by API, to indicate that there are more results available.
* A response header **'X-TOTAL-COUNT'** will be made available by API, to indicate total number of records in the result.

### ***Sample Use case***

To build a use case example, let’s say an API has 100 records when fetched without using any pagination params, then following scenarios are possible with pagination params –

* when called with */rest/<List API>?page=1&size=10*
  + will fetch records from 1 to 10 **with** response header 'HAS\_MORE\_RECORDS'
  + 'X-TOTAL-COUNT' will be set to 100
* when called with */rest/<List API>?page=3&size30*
  + will fetch records from 61 to 90 **with** response header 'HAS\_MORE\_RECORDS'
  + 'X-TOTAL-COUNT' will be set to 100
* when called with */rest/<List API>?page=4&size30*
  + will fetch records from 91 to 100 **without** response header 'HAS\_MORE\_RECORDS'
  + 'X-TOTAL-COUNT' will be set to 100
* when called with */rest/<List API>?page=5&size30*
  + will return HTTP 400 response
  + 'X-TOTAL-COUNT' will be set to 100

## ***Filter***

### ***Overview***

Like Pagination, Filter query params are enabled in selected List APIs only, which is indicated in their respective swagger documentation. Filtration capability allows callers to associate response fields with filters like Equals, StartsWith, etc., or to chain more than one filter using AND filter.

Usage instructions of filters can be summarized in following steps –

1. For a List API, which supports filters, create a filter json object, following supported filter schema.
2. Pick the fields of List API, which supports that filter type and create filter json object.
3. Use any JSON minification tool to minify the filter json object.
4. Use any URL encoding tool to do URL encoding of minified JSON object.
5. Pass URL encoded filter expression as ‘**filter**’ query param to the List API.

### ***Schema***

Overall call to pass a filter parameter is as simple as passing ‘***filter***’ query parameter to filterable list API. Like –

*/rest/<List API>?****filter****=<URL encoded filter expression>*

*URL encoded filter expression* is the URL encoded form of filter JSON object. A filter object has a **type** field, which accepts any of the predefined filter types and other fields are determined by type of filter used.

#### **Supported Filters Schema**

Following filter types are supported in Horizon Server REST APIs –

* ‘**Equals**’ - Single value field filter.
* ‘**NotEquals**’ - Single value field filter.
* ‘**Contains**’ – Multi value field filter.
* ‘S**tartsWith**’ - Single value field filter.
* ‘**Between**’ - Range based field filter.
* ‘**Not**’ – a logical Not filter on any of the other filter.
* ‘**And**’ – Chain filter to group more than one filters of other types.
* ‘**Or**’ - Chain filter to group more than one filters of other types.

##### *Single value field filter schema*

All single value filter (Equals, NotEquals, Contains, StartsWith) have following object schema –

**{**

**“type”: <filter type>,**

**“name”: <field name>,**

**“value”: <value of field to be used for filter>**

**}**

For example –

{

"type":"Equals",

"name":"domain",

"value":"ad-example0"

}

##### *Multi value filter schema*

Multi value filter (Contains) has following object schema –

**{**

**“type”: <filter type>,**

**“name”: <field name>,**

**“value”: [<value of field>, <value of field>]**

**}**

For example –

{

"type":"Equals",

"name":"domain",

"value":["ad-example0", “ad-example1”]

}

##### *Range filter schema*

Range filter (Between) has following object schema –

**{**

**“type”: ”Between”,**

**“name”: <field name>,**

**“fromValue”: <value of field to be used for filter>,**

**“toValue”: <value of field to be used for filter>**

**}**

For example –

{

"type":"Between",

"name":"assigned\_users",

"fromValue":"10",

"toValue":"20"

}

##### *Not filter schema*

A ‘Not’ filter is a logical NOT operation on any one of the predefined filter objects with following object schema –

**{**

**“type”: ”Not”,**

**“filter”: <filter object>**

**}**

For example –

{

"type":"Not",

"filter": {

"type":"Equals",

"name":"domain",

"value":"ad-example0"

}

}

##### *Chain filter schema*

Chain filters (And, Or) are logical AND/OR operations on more than one of the predefined filters objects with following object schema –

**{**

**“type”: <filter type>,**

**“filters”: [<filter object>, <filter object>]**

**}**

For example –

{

"type":"And",

"filters": [

{

"type":"Equals",

"name":"domain",

"value":"ad-example0"

},

{

"type":"StartsWith",

"name":"name",

"value":"test"

}

]

}

#### **URL encoding**

Once the filter JSON object is formed, *it is recommended to minify the JSON before doing the URL encoding*.

User can use any online or offline editor-based JSON minification tool to minify the JSON.

Post minification, JSON needs to be URL encoded (**using UTF-8 encoding only**) using any of the online or offline tools based on user preference.

For example,

1. If we have following filter object which is to be sent in as **filter** query param to API –

{

"type":"And",

"filters": [

{

"type":"Equals",

"name":"domain",

"value":"example"

},

{

"type":"StartsWith",

"name":"name",

"value":"test"

}

]

}

1. Its minified form will be –

{"type":"And","filters":[{"type":"Equals","name":"domain","value":"example"},{"type":"StartsWith","name":"name","value":"test"}]}

1. Its URL encoded form will be –

%7B%22type%22%3A%22And%22%2C%22filters%22%3A%5B%7B%22type%22%3A%22Equals%22%2C%22name%22%3A%22domain%22%2C%22value%22%3A%22example%22%7D%2C%7B%22type%22%3A%22StartsWith%22%2C%22name%22%3A%22name%22%2C%22value%22%3A%22test%22%7D%5D%7D

1. And API call would be like –

*/rest/<ListAPI>?filter=%7B%22type%22%3A%22And%22%2C%22filters%22%3A%5B%7B%22type%22%3A%22Equals%22%2C%22name%22%3A%22domain%22%2C%22value%22%3A%22example%22%7D%2C%7B%22type%22%3A%22StartsWith%22%2C%22name%22%3A%22name%22%2C%22value%22%3A%22test%22%7D%5D%7D*

### ***API Fields supporting filters***

As mentioned before, only selected List APIs are enabled with Filtration capabilities and are duly indicated in swagger API reference.

Similarly, for any filterable List API, response fields may support different filters on them. Swagger API reference of response model will indicate fields which supports filters and will also indicate which filters that respective field supports.

Refer to Swagger API reference for picking the right field name when building field filter schema for any filter type.

## ***Using generated client code from spec (from 8.1)***

When using a generated client from swagger spec, eligible list APIs with Pagination and Filter capabilities can be used from client code.

Pagination params of ‘page’ and ‘size’ can be simply passed as ‘integer’ values.

Filter schema as discussed in earlier sections is generated into model class for each filter type. These filter model objects needs to serialized as JSON and passed as ‘String’ input to filter eligible APIs.

### ***Generating client code***

We can use any clientgen library which accepts swagger spec (Open API Spec v2) as input.

For example, to generate Java client using swagger-codegen-cli library using ‘rest-api-swagger-docs.json’ file, one can use following command –

java -jar swagger-codegen-cli.jar generate --api-package 'com.omnissa.vdi.rest.client.api' --model-package 'com.omnissa.vdi.rest.client.model' --artifact-version 8.1 --group-id 'com.omnissa.vdi' --artifact-id rest-client -i rest-api-swagger-docs.json -l java -o REST\_API\_client

Now one can import REST\_API\_client project as maven/gradle project and start using the client (One has to still prepare client code for SSL validation and set the apikey as Bearer <access token> from login api)

### ***Using Pagination and Filter in Client code***

As categorized in spec, generated client code categorizes each API endpoint in respective category source file. For example, ‘listADUserOrGroupSummary’ is available as part of ExternalApi (these might be different with different clientgen library).

To call ‘listADUserOrGroupSummary’ with Pagination and filter, one use following example for reference –

Integer page = 1;

Integer size = 10;

AndFilter andFilter = new AndFilter();

andFilter.setType(BaseFilter.TypeEnum.AND);

EqualsFilter equalsFilter = new EqualsFilter();

equalsFilter.setName("domain");

equalsFilter.setValue("example.com");

equalsFilter.setType(BaseFilter.TypeEnum.EQUALS);

andFilter.addFiltersItem(equalsFilter);

Gson gson = new GsonBuilder().create();

String filterStr = gson.toJson(andFilter);

List<ADUserOrGroupSummary> response = externalApi.listADUserOrGroupSummary(filterStr, false, page, size);

Two important points to be noted when using Filter classes in generated client are –

* We need to pass filter type using setType() method of each filter class, passing the same corresponding BaseFilter.TypeEnum value as that of filter class.
* Effective filter string passed into API should be JSON serialized using any json library.

Like this, any of the filter class, supported on the filterable APIs can be used.

## ***Sorting***

### ***Overview***

Like Pagination, Sorting query params are enabled in selected List APIs only, which is indicated in their respective swagger documentation. Sorting capability allows callers to sort result in ascending as well as descending order.

Following key points needs to be considered for using Sorting –

* Any Sorting enabled List API will be capable of receiving two **optional** query params to support sorting – **sort\_by**  and **order\_by**.
* If sorting is intended, then sort\_by is mandatory parameter.
* order\_by is an optional parameter and can be set to either ASC for ascending order or DESC for descending order. If it is not present then result is sorted in ascending order.

### ***API Fields supporting sorting***

As mentioned before, only selected List APIs are enabled with Sorting capabilities and are duly indicated in swagger API reference.

Similarly, for any sortable List API, response fields may support sorting by them. Swagger API reference of response model will indicate fields which supports sorting.

Refer to Swagger API reference for picking the right field to be used for sort\_by query param.

### ***Sample Use case***

To build a use case example, let’s say an API has 4 records, with name fields as apple, ball, cat and dog and supporting sorting for name field, then following scenarios are possible with sorting params –

* when called with */rest/<List API>?sort\_by=name*
  + will fetch records in order apple, ball, cat and dog.
* when called with */rest/<List API>?sort\_by=name*&order\_by=ASC
  + will fetch records in order apple, ball, cat and dog.
* when called with */rest/<List API>?sort\_by=name*&order\_by=DESC
  + will fetch records in order dog, cat, ball, apple.
* when called with */rest/<List API>?*order\_by=ASC
  + will return HTTP 400 response