



# Denodo OData Service - User Manual

Revision 20150730

## NOTE

This document is confidential and proprietary of **Denodo Technologies**.  
No part of this document may be reproduced in any form by any means without prior written authorization of **Denodo Technologies**.

Copyright © 2015  
Denodo Technologies Proprietary and Confidential

## CONTENTS

<b>1 OVERVIEW.....</b>	<b>3</b>
<b>2 INSTALLATION.....</b>	<b>4</b>
<b>3 FEATURES.....</b>	<b>5</b>
<b>4 SERVING METADATA.....</b>	<b>6</b>
<b>5 QUERYING DATA: THE BASICS.....</b>	<b>8</b>
5.1 QUERYING COLLECTIONS.....	8
5.2 OBTAINING ENTRIES BY PRIMARY KEY.....	8
5.3 ACCESSING INDIVIDUAL PROPERTIES.....	8
5.4 ACCESSING INDIVIDUAL PROPERTY VALUES.....	9
5.5 ACCESSING COMPLEX PROPERTIES.....	9
5.6 COUNTING ELEMENTS IN A COLLECTION: \$COUNT.....	10
5.7 ESTABLISHING RESPONSE FORMAT: ATOMPUB VS JSON.....	10
<b>6 NAVIGATING ASSOCIATIONS.....</b>	<b>12</b>
6.1 PUBLISHING VDP ASSOCIATIONS THROUGH ODATA.....	12
6.2 QUERYING ASSOCIATED ENTRIES.....	13
6.3 LISTS OF LINKS BETWEEN ENTRIES.....	14
<b>7 SELECTION, PROJECTION AND ORDERING.....</b>	<b>15</b>
7.1 SELECTION: \$FILTER.....	15
7.2 PROJECTION: \$SELECT.....	16
7.3 ORDERING RESULTS: \$ORDERBY.....	17
<b>8 PAGINATION.....</b>	<b>18</b>
8.1 SPECIFYING MAXIMUM NUMBER OF RESULTS / PAGE SIZE: \$TOP.....	18
8.2 SPECIFYING OFFSET: \$SKIP.....	19
8.3 ASKING FOR TOTAL RESULT COUNT: \$INLINECOUNT.....	19
8.4 DEFAULT PAGINATION MECHANISM.....	21
<b>9 LIMITATIONS.....</b>	<b>22</b>

## 1 OVERVIEW

---

[OData](#) (**Open Data Protocol**) is an OASIS standard. It is an open protocol for building and consuming RESTful APIs. It means that OData enables the creation of HTTP-based data services, which allow resources identified using Uniform Resource Identifiers (URIs) and defined in a data model, to be published and edited by Web clients using simple HTTP messages.

Denodo OData Service allows Denodo users to connect to the Denodo Platform and query its databases using an [OData 2.0](#) interface.

More on the OData standard here: <http://www.odata.org>

## 2 INSTALLATION

---

The Denodo OData Service distribution consists of:

- A war file (denodo-odata2-service-5.5.war)
- A documentation folder containing this user manual (/doc folder)

The war file has a configuration file in the WEB-INF/classes folder called `configuration.properties`:

```
vdp.host=localhost  
vdp.port=9999  
odataserver.address=/denodo-odata.svc  
server.pageSize=1000
```

This file allows changing some properties:

- `vdp.host` the Denodo Virtual DataPort server to connect to
- `vdp.port` the Denodo Virtual DataPort server port to connect to
- `odataserver.address` Denodo OData server address
- `server.pageSize` default number of returned entries per request (see **Default pagination mechanism section** for more information)

For running the Denodo OData Service you need to deploy the war file in a Java web application container. **Apache Tomcat 7+** (using Java 7 or later) is recommended.

Once you deployed the war you can use the Denodo OData Service from a web client, using HTTP Basic Authentication with VDP-valid credentials). Therefore you may use URLs that are of the form:

**`http://localhost:8080/denodo-odata.svc/<DBNAME>`**

(Note that for the sake of this document, we will consider the OData server to be installed at the ROOT context of the web server)

## 3 FEATURES

---

Denodo OData Service provides the following main features:

- **Read-only** access to Denodo databases via **OData 2.0**.
- Show metadata of the Denodo database to which the connection is made.
- Address collections.
- Address entries.
- Address properties of an entry.
- Address property values.
- Address links between entries.
- Format results both in AtomPub and/or in JSON (\$format)
- Query string options
  - \$filter
  - \$select
  - \$orderby
- Pagination: \$skip, \$top, \$inlinecount

## 4 SERVING METADATA

---

There are two types of metadata documents:

- The **Service Document** that lists all the top-level feeds, which are collections of typed Entries and it is available at the Service Root URI, the root of the OData Service, specifying the database name where we are going to get information, `denodo-odata.svc/<DBNAME>`. Below, there is an example where the accessible collections of movies database are actor, address, city, country, film and film\_actor.

**`http://localhost:8080/denodo-odata.svc/movies`**

```
<?xml version='1.0' encoding='utf-8'?>
<service          xml:base="http://localhost:8080/denodo-odata.svc/movies/"
  xmlns="http://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/Atom">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="actor">
      <atom:title>actor</atom:title>
    </collection>
    <collection href="address">
      <atom:title>address</atom:title>
    </collection>
    <collection href="city">
      <atom:title>city</atom:title>
    </collection>
    <collection href="country">
      <atom:title>country</atom:title>
    </collection>
    <collection href="film">
      <atom:title>film</atom:title>
    </collection>
    <collection href="film_actor">
      <atom:title>film_actor</atom:title>
    </collection>
  </workspace>
</service>
```

- The **Service Metadata Document**, also called **Entity Data Model (EDM)**, that describes the data model exposed in **CSDL** (Common Schema Definition Language), an application of XML. This Entity Data Model is available at the `.../$metadata` URL and includes five types of structures:

- Entity Types (also “feeds” or “collections”)
- Associations
- Entity Sets
- Association Sets
- Imported Functions

**`http://localhost:8080/denodo-odata.svc/movies/$metadata`**

---

Example of feed:

```
<EntityType Name="actor">
  <Key>
    <PropertyRef Name="actor_id"/>
  </Key>
  <Property Name="actor_id" Type="Edm.Int16" Nullable="false"/>
  <Property Name="first_name" Type="Edm.String" Nullable="true" MaxLength="45"/>
  <Property Name="last_name" Type="Edm.String" Nullable="true" MaxLength="45"/>
  <Property Name="last_update" Type="Edm.DateTimeOffset" Nullable="true"
    Precision="19"/>
</EntityType>
```

Example of association that represents a relationship between country and city. Every country element is related with zero or more city elements:

```
<Association Name="country_city">
  <Documentation>
    <Summary>Association between cities and countries</Summary>
  </Documentation>
  <End Type="com.denodo.odata2.country" Multiplicity="1" Role="country"/>
  <End Type="com.denodo.odata2.city" Multiplicity="*" Role="cities"/>
  <ReferentialConstraint>
    <Principal Role="country">
      <PropertyRef Name="country.country_id"/>
    </Principal>
    <Dependent Role="cities">
      <PropertyRef Name="city.country_id"/>
    </Dependent>
  </ReferentialConstraint>
</Association>
```

The Denodo OData Service maps these OData structures to VDP concepts like this:

Denodo ODATA Service	VDP
<b>Entity Type</b>	<b>View Definition</b>
<i>Entity Type &gt; Property</i>	<i>View Column</i>
<i>Entity Type &gt; Navigation Property</i>	<i>Association Role</i>
<b>Association</b>	<b>Association Definition</b>
<b>Entity Set</b>	<b>View Data</b>
<b>Association Set</b>	<b>(Associated data)</b>
<b>Imported Functions</b>	-

## 5 QUERYING DATA: THE BASICS

---

### 5.1 QUERYING COLLECTIONS

In the Service Metadata Document (see the **Metadata section** for more information) you can see the entity set names and to see its data you must use the following URL:

```
/denodo-odata.svc/<DBNAME>/collectionName
```

Example (note that, for the sake of simplicity, we are removing the server and port from the example):

```
/denodo-odata.svc/movies/actor
```

### 5.2 OBTAINING ENTRIES BY PRIMARY KEY

Each entry could be identified using its primary key property:

```
denodo-odata.svc/<DBNAME>/collectionName(keyvalue)
```

Examples:

```
/denodo-odata.svc/movies/actor(1)  
/denodo-odata.svc/movies/store_category('F0')
```

The PK can be a compound key, and in this case you must include all values separated by commas:

```
/denodo-odata.svc/<DBNAME>/collectionName(key1, key2)
```

Example:

```
/denodo-odata.svc/movies/film_actor(actor_id=1, film_id=1)
```

Note that, when there is not a defined primary key, this option is unavailable.

### 5.3 ACCESSING INDIVIDUAL PROPERTIES

Properties of an entry can be accessed individually:



```
/denodo-odata.svc/<DBNAME>/collectionName(key)/propertyName
```

Example:

```
/denodo-odata.svc/movies/actor(1)/first_name
```

Response:

```
{
  "d": {
    "first_name": "PENELOPE"
  }
}
```

## 5.4 ACCESSING INDIVIDUAL PROPERTY VALUES

The value of a property is available as a raw value:

```
/denodo-odata.svc/<DBNAME>/collectionName(key)/propertyName/$value
```

Example:

```
/denodo-odata.svc/movies/actor(1)/first_name/$value
```

Response:

PENELOPE

## 5.5 ACCESSING COMPLEX PROPERTIES

Properties can be complex but they are also accessible. You must point out the property path, from the complex to the simple one:

```
/denodo-odata.svc/<DBNAME>/collectionName(pk)/propName/complexProp/propName
```

For example, for the following `film_data` complex field in a `struct_table_film` entity:

```
"table_id": "1",
"film_data": {
  "__metadata": {
    "type": "com.denodo.odata2.struct_table_film_film_data"
  },
  "id": "1",
  "title": "ACADEMY DINOSAUR",
  "description": "ELIZABETH"
```

```
}
```

...we could perform the following call:

```
/denodo-odata.svc/movies/struct_table_film(1)/film_data/title
```

Response:

```
{
  "d": {
    "title": "ACADEMY DINOSAUR"
  }
}
```

## 5.6 COUNTING ELEMENTS IN A COLLECTION: \$COUNT

If you want to know the number of elements (entries) in a collection you have to add \$count to the URL:

```
/denodo-odata.svc/<DBNAME>/collectionName/$count
```

Example:

```
/denodo-odata.svc/movies/actor/$count
```

Response:

```
200
```

### 5.6.1

## 5.7 ESTABLISHING RESPONSE FORMAT: ATOMPUB VS JSON

OData resources can be represented in AtomPub (XML) or JSON. Using the system query option \$format you might specify the format to be used by the OData service to display the response.

The example URLs below will show the same data but represented using the AtomPub format as defined in [OData:Atom](#) in the first one and in the second using the JSON format as defined in [OData:JSON](#).

```
/denodo-odata.svc/movies/actor?$format=atom
```

```
/denodo-odata.svc/movies/actor?$format=json
```

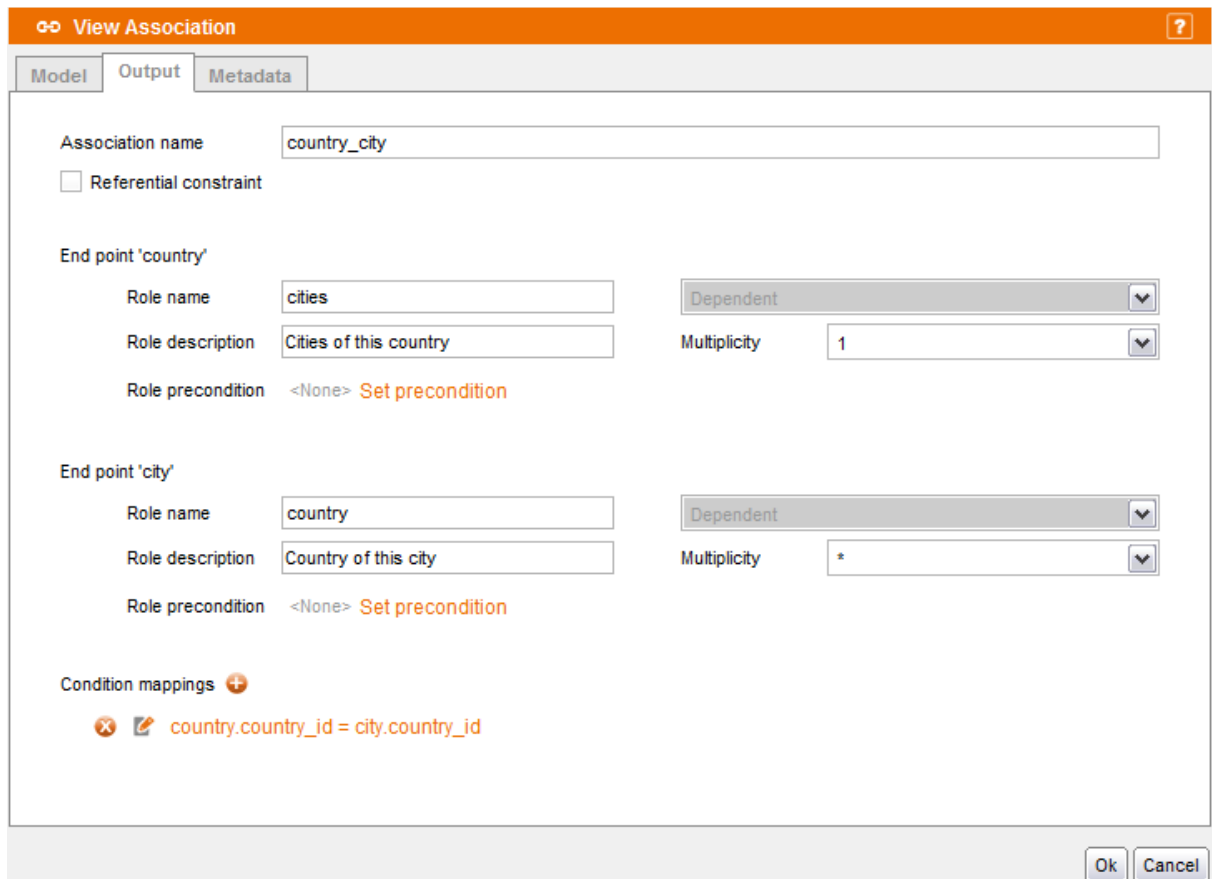
An alternative and equally valid way to request a JSON response from the OData Service (note that AtomPub is the default) is by means of specifying an Accept HTTP header in the request, such as:

Accept: application/json

## 6 NAVIGATING ASSOCIATIONS

### 6.1 PUBLISHING VDP ASSOCIATIONS THROUGH ODATA

Denodo Virtual DataPort allows you to define relationships between the elements of two views. The following example shows an association where the elements of the `country` view can be related with the elements of the `city` view. Every country is related with zero or more cities.



The association can be seen in the Service Metadata Document:

```
<Association Name="country_city">
  <Documentation>
    <Summary></Summary>
  </Documentation>
  <End Type="com.denodo.odata2.country" Multiplicity="1" Role="country"/>
  <End Type="com.denodo.odata2.city" Multiplicity="*" Role="cities"/>
  <ReferentialConstraint>
    <Principal Role="country">
```

```

        <PropertyRef Name="country.country_id"/>
    </Principal>
    <Dependent Role="cities">
        <PropertyRef Name="city.country_id"/>
    </Dependent>
</ReferentialConstraint>
</Association>

```

## 6.2 QUERYING ASSOCIATED ENTRIES

When there are associations between entities defined in VDP, it is possible to navigate those associations in order to get all the entries associated with a particular entry. This is done by means of **navigation properties**, a special kind of property created in OData for this specific purpose (allowing the navigation of declared associations).

```
/denodo-odata.svc/<DBNAME>/collectionName(key)/navigationPropertyName
```

Example, being `cities` a navigation property in the `country` entity type (VDP view) that navigates an association towards the `city` entity type (another VDP view).

```
/denodo-odata.svc/movies/country(1)/cities
```

Response:

```

{
  "d": {
    "results": [
      {
        "__metadata": {
          "id": "http://localhost:8080/denodo-odata.svc/movies/city(251)",
          "uri": "http://localhost:8080/denodo-odata.svc/movies/city(251)",
          "type": "com.denodo.odata2.city"
        },
        "city_id": 251,
        "city": "Kabul",
        "country_id": 1,
        "last_update": "\\Date(1139975125000+0060)\\",
        "country": {
          "__deferred": {
            "uri": "http://localhost:8080/denodo-
odata.svc/movies/city(251)/country"
          }
        }
      }
    ]
  }
}

```

### 6.3 LISTS OF LINKS BETWEEN ENTRIES

You can get the links to the collection of entries relationed with one in particular:

```
/denodo-odata.svc/<DBNAME>/collectionName(key)/$links/navProp
```

The URL specified above will show all the URLs which allows the access to every entry associated with the entry collectionName(key) through the navigation property navProp.

Example:

```
/denodo-odata.svc/movies/country(2)/$links/cities
```

Response:

```
{
  "d": [
    {
      "uri": "http://localhost:8080/denodo-odata.svc/movies/city(59)"
    },
    {
      "uri": "http://localhost:8080/denodo-odata.svc/movies/city(63)"
    },
    {
      "uri": "http://localhost:8080/denodo-odata.svc/movies/city(483)"
    }
  ]
}
```

## 7 SELECTION, PROJECTION AND ORDERING

The data returned by the OData service can be controlled to some extent. To achieve this management there are several system query options that are prefixed with a \$ character.

### 7.1 **SELECTION: \$FILTER**

A URI with a \$filter system query option identifies a subset of the entries from the collection of entries identified by the resource path section of the URI. This subset must satisfy the predicate expression specified by the query option.

The filter query option supports references to properties and literals. The latter can be strings (enclosed in single quotes), numbers and boolean values.

In addition to the logical operators (see the **Logical operators section**) and the string functions (see the **String functions section**) explained below, you can use parenthesis to denote precedence by grouping.

#### 7.1.1.1 Logical operators

Operator	Description	Example
Eq	Equal	/actor?\$filter=first_name eq 'GRACE'
Ne	Not equal	/actor?\$filter=first_name ne 'GRACE'
Gt	Greater than	/actor?\$filter=actor_id gt 5
Ge	Greater than or equal	/actor?\$filter=actor_id ge 5
Lt	Less than	/actor?\$filter=actor_id lt 10
Le	Less than or equal	/actor?\$filter=actor_id le 10
And	Logical and	/actor?\$filter=actor_id gt 5 and actor_id lt 10
Or	Logical or	/actor?\$filter=actor_id lt 5 or first_name eq 'GRACE'
Not	Logical negation	/actor?\$filter=not (actor_id eq 1)

#### 7.1.1.2 String functions

There are three string functions available that allow managing strings:

- `substringof(string p0, string p1)` returns true when the value of the property name specified in p1 contains the string p0. Otherwise returns false.
- `startswith(string p0, string p1)` returns true when the value of the property name specified in p0 starts with the string p1. Otherwise returns false.
- `indexof(string p0, string p1)` returns the position of the string p1 in the value of the property name specified in p0.

The following table shows a summary and examples of this functions:

Function	Example
<code>bool substringof(string p0, string p1)</code>	<code>/actor?\$filter=substringof('LO', first_name) eq true</code>
<code>bool startswith(string p0, string p1)</code>	<code>/actor?\$filter=startswith(first_name, 'JO') eq true</code>
<code>int indexof(string p0, string p1)</code>	<code>/actor?\$filter=indexof(last_name, 'LO') eq 3</code>

## 7.2 **PROJECTION: \$SELECT**

A data service URI with a \$select system query option specifies that a response from the service should return a subset of the properties that are written as a comma-separated list of selection clauses. Each selection clause may be a property name or the \* character, which will show all the properties for the resource determined by the resource path section of the URI. Also note that complex properties are forbidden as selection clauses.

Example:

```
/denodo-odata.svc/movies/actor?$select=actor_id,first_name,last_name
```

Response:

```
...
{
  "__metadata": {
    "id": "http://localhost:8080/denodo-odata.svc/movies/actor(1)",
    "uri": "http://localhost:8080/denodo-odata.svc/movies/actor(1)",
    "type": "com.denodo.odata2.actor"
  },
  "actor_id": 1,
  "first_name": "PENELOPE",
  "last_name": "GUINESS"
},
...
```



Another example:

```
/denodo-odata.svc/movies/actor?$select=*
```

Response:

```
...
{
  "__metadata": {
    "id": "http://localhost:8080/denodo-odata.svc/movies/actor(1)",
    "uri": "http://localhost:8080/denodo-odata.svc/movies/actor(1)",
    "type": "com.denodo.odata2.actor"
  },
  "actor_id": 1,
  "first_name": "PENELOPE",
  "last_name": "GUINNESS",
  "last_update": "\\Date(1139974473000+0060)\\/"
},
...
```

### 7.3 ORDERING RESULTS: \$ORDERBY

If the URL contains the \$orderby query string option it specifies the attributes that are used to order the collection of entries identified by the resource path section of the URI:

```
/denodo-odata.svc/<DBNAME>/collectionName?$orderby=attribute [asc|desc]
```

To order the collection the resource path must identified a collection of entries, otherwise this option is unavailable.

The optional keywords asc and desc determine the direction of the sort (ascending or descending, respectively). Default value is asc, therefore sorting will be ascending if you do not use one of these keywords.

You can also sort by multiple attributes:

```
/denodo-odata.svc/<DBNAME>/collectionName?$orderby=attribute1 [asc|desc],attribute2 [asc|desc]
```

Example:

```
/denodo-odata.svc/movies/address?$orderby=zip,client_idenfifier desc
```

## 8 PAGINATION

---

### 8.1 SPECIFYING MAXIMUM NUMBER OF RESULTS / PAGE SIZE: \$TOP

With the \$top option you can select the n first entries of the collection determined by the resource path section of the URI:

```
/denodo-odata.svc/<DBNAME>/collectionName?$top=n
```

Note that n is a positive integer, a negative value means that the URL is malformed.

Example:

```
/denodo-odata.svc/movies/address?$top=1
```

Response:

```
{
  "d": {
    "results": [
      {
        "__metadata": {
          "id": "http://localhost:8080/denodo-odata.svc/movies/address('C001')",
          "uri": "http://localhost:8080/denodo-odata.svc/movies/address('C001')",
          "type": "com.denodo.odata2.address"
        },
        "client_identifier": "C001",
        "street": "3989 Middlefield Rd",
        "city": "San Jose",
        "zip": "94085",
        "state": "CA",
        "primary_phone": "(408) 813-9318",
        "country": "UNITED STATES"
      }
    ],
    "__next": "http://localhost:8080/denodo-odata.svc/movies/address?$top=1&$skiptoken=1"
  }
}
```

This query string option also allows pagination (see the **Pagination** section for more information).

## 8.2 SPECIFYING OFFSET: \$SKIP

With the option \$skip, the n first entries of the collection identified by the resource path section of the URI not will be shown in the response:

denodo-odata.svc/<DBNAME>/collectionName?\$skip=n

Note that n is a positive integer, a negative value means that the URL is malformed.

Example:

/denodo-odata.svc/movies/address?\$skip=79

Response:

```
{
  "d": {
    "results": [
      {
        "__metadata": {
          "id": "http://localhost:8080/denodo-odata.svc/movies/address('C080')",
          "uri": "http://localhost:8080/denodo-odata.svc/movies/address('C080')",
          "type": "com.denodo.odata2.address"
        },
        "client_identifier": "C080",
        "street": "2347 Santa Ana St",
        "city": "Palo Alto",
        "zip": "94303-3141",
        "state": "CA",
        "primary_phone": "(650) 856-9738",
        "country": "UNITED STATES"
      }
    ]
  }
}
```

## 8.3 ASKING FOR TOTAL RESULT COUNT: \$INLINECOUNT

The number of entries in a collection identified by the resource path section of the URI may be added to the data showed in the response using the \$inlinecount system query option. This option allow two possible values:

- allpages the OData server include a count of number of entities in the collection identified by the URI.
- none the OData server do not include a count in the response. This option is equivalent to a URI without the \$inlinecount option.

Note that the count is calculated after applying any [\\$filter system query options](#) present in the URI.

Examples:

```
/denodo-odata.svc/movies/actor?$inlinecount=allpages
```

Response:

```
{
  "d": {
    "__count": "200",
    "results": [
      ...
    ]
  }
}
```

Another example:

```
/denodo-odata.svc/movies/actor?$inlinecount=allpages&$filter=actor_id
eq 1
```

Response:

```
{
  "d": {
    "__count": "1",
    "results": [
      {
        "__metadata": {
          "id": "http://localhost:8080/denodo-odata.svc/movies/actor(1)",
          "uri": "http://localhost:8080/denodo-odata.svc/movies/actor(1)",
          "type": "com.denodo.odata2.actor"
        },
        "actor_id": 1,
        "first_name": "PENELOPE",
        "last_name": "GUINNESS",
        "last_update": "\\Date(1139974473000+0060)\\/"
      }
    ]
  }
}
```

Another example:

```
/denodo-odata.svc/movies/actor?$inlinecount=none
```

Response:

*(Actor data, just like without \$inlinecount option.)*

## 8.4 DEFAULT PAGINATION MECHANISM

Whenever the Denodo OData Service has to return a collection of entries which size exceeds that configured at the server `.pageSize` property of its configuration file, it will split the response into pages, returning only the first 1000 entries (the default value for page size).

Also, both in this case (default pagination) and in the case that a specific pagination configuration is specified by the client by means of a `$top + $skip` combination (see above), the OData Service will add to the response a link called `__next`, which will easily allow the client to request the next page of results.

URIs in `__next` links add a `$skiptoken` parameter to the original URL. They look like this:

```
/denodo-odata.svc/movies/actor?$skiptoken=1
```

Or, for example, if we had specified a max page size different to the default one (with `$top`):

```
/denodo-odata.svc/movies/actor?$top=2
```

...we would get:

```
...
"__next":                "http://localhost:8080/denodo-odata.svc/movies/actor?
$top=2&$skiptoken=1"
...
```

If the original request includes a `$skip` query string option in addition to `$top`, the result will show the pagination link with these two options and a `$skiptoken` as in the examples above. Each page of results will start in the entry whose position is given by `skip_value + top_value * skiptoken`.

Example request:

```
/denodo-odata.svc/movies/actor?$top=2&$skip=1
```

Response:

```
...
"__next":                "http://localhost:8080/denodo-odata.svc/movies/actor?
$top=2&$skip=1&$skiptoken=1"
...
```

## 9 LIMITATIONS

---

### Read-only access

The access to Denodo databases via this Denodo OData Service is read-only.

### Representing arrays

OData 2.0 does not support Collections/Bags/Lists that will allow us to give support to arrays as complex objects. Now they are displayed between square brackets as a comma-separated list of values.

### Unavailable operators in the \$filter system query option

Denodo OData Service does not support arithmetic operators: add (addition), sub (subtraction), mul (multiplication), div (division) and mod (modulo).

### Unavailable functions in the \$filter system query option

Denodo OData Service provides support for substringof, startswith and indexof, but there are several string functions that it does not support:

- bool endswith(string p0, string p1)
- int length(string p0)
- int indexof(string p0, string p1)
- string replace(string p0, string find, string replace)
- string substring(string p0, int pos)
- string substring(string p0, int pos, int length)
- string tolower(string p0)
- string toupper(string p0)
- string trim(string p0)
- string concat(string p0, string p1)

Denodo OData Service does not provide support for date functions:

- int day(DateTime p0)
- int hour(DateTime p0)
- int minute(DateTime p0)
- int month(DateTime p0)
- int second(DateTime p0)
- int year(DateTime p0)

Denodo OData Service interface does not provide support for math functions:

- double round(double p0)

- decimal round(decimal p0)
- double floor(double p0)
- decimal floor(decimal p0)
- double ceiling(double p0)
- decimal ceiling(decimal p0)

Denodo OData Service does not provide support for type functions:

- bool IsOf(type p0)
- bool IsOf(expression p0, type p1)

### Unavailable \$expand system query option

This system query option indicates that entries associated with the elements identified by the resource path section must be represented inline but it is not supported by Denodo OData Service.

### Navigation properties in the select system query option

Denodo OData Service does not allow navigation properties as selection clauses.

### Filtering by datetime

It is not available the filter URI for search by datetime as in the example below:

[http://localhost:8080/denodo-odata.svc/movies/actor?\\$filter==last\\_update\\_datetime'2016-06-06'](http://localhost:8080/denodo-odata.svc/movies/actor?$filter==last_update_datetime'2016-06-06') It

### Navigation using complex properties

When there is an association where one of the elements of an end point is a field of a register, Denodo Virtual DataPort does not allow the navigation from the end point with the complex property.

### Order by using complex properties

Denodo Virtual DataPort does not support the order by clause using fields of registers, therefore the \$orderby query string option is not available if you want to sort by a field of a complex property

### Counting entries of a collection when there is navigation

Navigational queries does not allow the count ( \* ) function, then if the URL contains a navigation the option of counting entries in a collection (\$count) is not available.

### Using the string function 'indexof' with complex properties

Denodo Virtual DataPort does not support this function when the property specified in the function is a field of a complex property.