Hitchhiker's Guide to the Output Module 5.7

Prerequisites

Knowledge of Collibra's API model is a prerequisite before diving into the query language offered by the Output Module.

Examples below are querying a REST API. Explaining how to execute REST calls is beyond the scope of this document.
Please note that this topic is widely documented online.

Terminology

Data Governance Center (DGC) uses an API model based upon the SBVR standard (Semantics of Business Vocabulary and Rules).
Over time, the user interface adopted a somewhat simpler terminology which is in line with the Data Governance concepts.
This means some of the entity names used in the Output Module at API level are different than the names used in the front-end.

Here is the list of the 6 entities that have been renamed. All other entity names are the same on both sides.

Frontend Terminology	API Level Terminology
Asset	Term
AssetType	ConceptType
Domain	Vocabulary
DomainType	VocabularyType
RelationType	BinaryFactType
Responsibility	Member

This documents will use both terminologies interchangeably, trying to give precedence to the frontend ones whenever possible.

In a nutshell

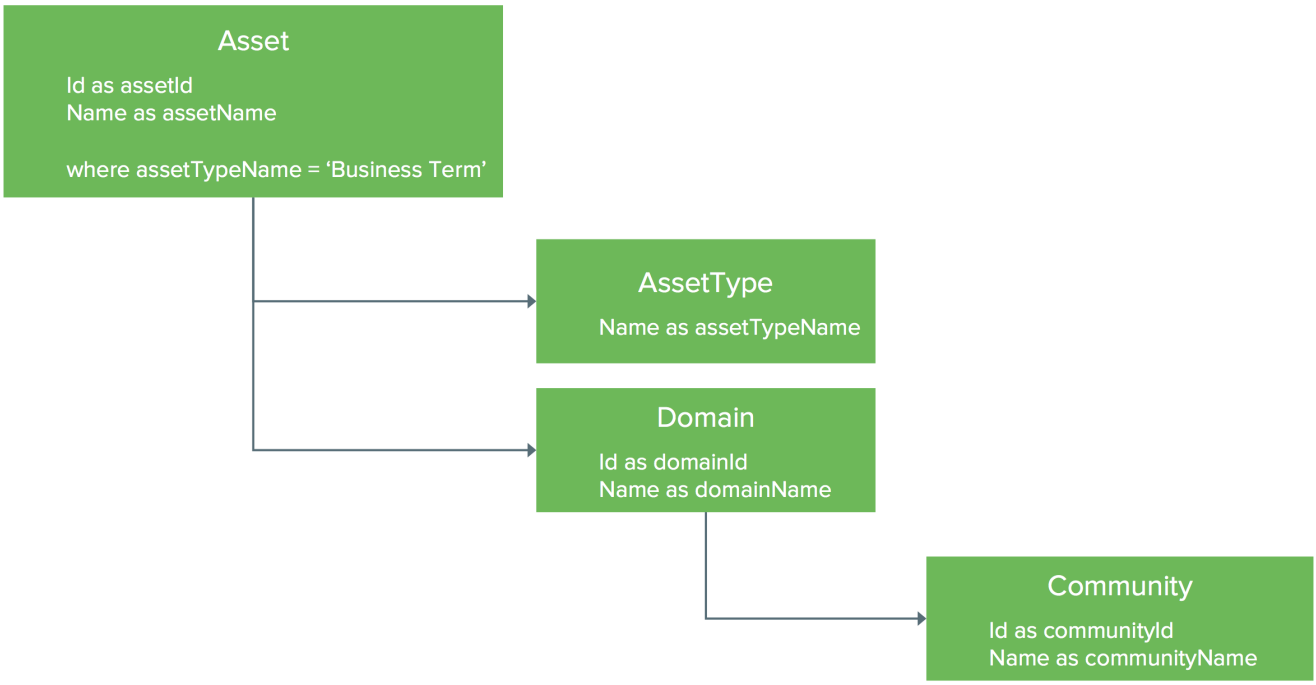
The Output Module is a lightweight **graph query engine** exposed through DGC's public API. It allows **different output formats** (json, xml, excel, csv) and provides a **one stop API to query most of DGC's entities** (Assets, communities, domains, types, ...) using **SQL-like filtering** capabilities. The query engine also **supports sorting** of entities using any of the available properties as well as **paging of the results**. Last but not least, the query engine also **ho nors view permissions** that were set for the particular logged in user that is issuing the REST call.

The Query Language

General Concepts

Given a set of well defined entities and relations (the API model), a user can create a query that takes the form of a single rooted tree graph and specify constraints that must hold true for any of the result nodes (I.e.: filtering the results).

Supposing we'd like to query all assets of type "Business Term" along with their respective domain and community, we'd need to specify the following tree graph:



Important concepts

- The graph is a single rooted tree graph. Multiple root nodes are not allowed, each node has only one parent
- For each of the selected properties, the user has to specify an alias that is unique within the graph query
- Filtering is specified on the node you want to filter and can reference any property of the current node or one of it's children, grandchildren, etc.
  In the above graph, Assets are filtered using their related AssetType name

Hello World Example

The format of the query language is JSON.

A full fledged query rapidly becomes overwhelming to read so we'll start with a very simple query and build up from there:

Select id and name for all communities as a flat list:

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" }
      }
    }
  }
}
```

The JSON object representing the query is called a **ViewConfig** as it defines a particular view (selection) on the data.

The JSON object containing the graph part of the query is called **Resources**. In this example, we just select the **Community** entity along with its **Id** and **Name** properties.

- Please note that **entities and properties' keys are case insensitive**, so "Community", "Id" could be written using any random casing
- This is **not true for other keys** like "ViewConfig", "Resources" or "name"
- Whenever a property is spelled out the same way as a reserved key word, one must use a different casing than the reserved key. E.g.: "name" (all lower case) cannot be used to reference a Community name.

Let's try

In order to test this API, a REST client is required (e.g.: Postman plugin for Chrome)

As mentioned earlier, a variety of different output formats are available. We'll first go with the format that resembles query the most: a JSON tree

Amongst the various endpoints available on the OutputView resource, we'll use the following:

{{domain}}/rest/latest/outputModule/export/json

Use a POST call with the following body:

```
{   "ViewConfig": {     "Resources": {       "Community": {         "Id": { "name": "communityId" },         "Name": { "name": "communityName" }       }     }   } }
```

Remember to set the content type header:

```
'Content-Type': 'application/json'
```

The output would be formatted as an array of communities:

```
{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "86a745f5-7e87-4851-a107-a3a272ccea0b",
        "communityName": "Second Community"
      }
    ]
  }
}
```

ViewConfig queries can be used with the following endpoints:

- {{domain}}/rest/latest/outputModule/export/{{xml | json}}
- {{domain}}/rest/latest/outputModule/export/{{xml | json}}-file
- {{domain}}/rest/latest/outputModule/export/{{xml | json}}-job

Adding Related Entities to the Tree

Let's take the previous query again and add the Users that have been assigned a Role at Community level.
In order to reach those entities, we need to get the Member entities which represent the assignments between a User, a Role and one of the following kind of resources: Asset, Domain or Community.

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" },
        "Member": {
          "User": {
            "Id": { "name": "userId" },
            "FirstName": { "name": "firstName" },
            "LastName": { "name": "lastName" }
          },
          "Role": {
            "Signifier": { "name": "roleName" }
          }
        }
      }
    }
  }
}
```

Navigating from one entity to another is a matter of **nesting** the entities. A formal list of available properties and relations for each entity follows later in this guide

The results would be formatted like so: (click on "Expand source" on the right)

```
{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "Member1": [
          {
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Admin"
              }
            ]
          },
          {
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Steward"
              }
            ]
          }
        ]
      }
    ]
  }
}
```


- With a ViewConfig the result tree always uses arrays for related entities, even when relations have a max cardinality of one.
  Each Member has max 1 User and max 1 Role, yet, arrays are returned. This might be a bit confusing if you're not aware of it.
- The result tree uses a generated entity alias in the response: Community0, Member1, User2, ...
  An index number is concatenated to the entity name in order to prevent any possible duplicate name in the JSON keys.
- The query engine knows about optional and required relations between entities, which is why we get "First Community" as a result even though there are currently no Users playing a Role in it. In other words, the relationship from Community to Member is optional.

Specifying your own alias for the entities

Auto generated aliases used in the response are not straightforward to use programmatically: Community0, Member1, User2
For this reason, the language allows specifying your own alias:

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "name": "community",
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" },
        "Member": {
          "name": "responsibility",
          "User": {
            "name": "employee",
            "Id": { "name": "userId" },
            "FirstName": { "name": "firstName" },
            "LastName": { "name": "lastName" }
          },
          "Role": {
            "name": "role",
            "Signifier": { "name": "roleName" }
          }
        }
      }
    }
  }
}
```

The results are then much more easy to parse:

```
{
  "view": {
    "community": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "responsibility": [
          {
            "employee": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "role": [
              {
                "roleName": "Admin"
              }
            ]
          },
          {
            "employee": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "role": [
              {
                "roleName": "Steward"
              }
            ]
          }
        ]
      }
    ]
  }
}
```


Adding a Related Entity More than Once

The previous query misses the Groups of Users that may play a Role in a given Community. So in order to get a better picture of who does what in our Communities, we'd need to also query the Groups that are linked through a Member for each Community.

Still building on the previous query, let's add **another relation** from Community to Member, this time selecting the related Groups.
Please note the Id property of the two Member nodes are also selected, this is purely for demonstration purposes.

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" },
        "Member": [
          {
            "Id": { "name": "userMemberId" },
            "User": {
              "Id": { "name": "userId" },
              "FirstName": { "name": "firstName" },
              "LastName": { "name": "lastName" }
            },
            "Role": {
              "Signifier": { "name": "userRoleName" }
            }
          },
          {
            "Id": { "name": "groupMemberId" },
            "Group": {
              "Id": { "name": "groupId" },
              "GroupName": { "name": "groupName" }
            },
            "Role": {
              "Signifier": { "name": "groupRoleName" }
            }
          }
        ]
      }
    }
  }
}
```

In order to be able to add the same related entity twice under the same node, all that is required is to **change the JSON object into an array of such objects**.
In this case, the "Member" JSON object became an array and the anonymous JSON objects composing the array are actually multiple Members.

Supposing we just added the "admin" Group to the second Community, the results would be formatted like so:

```
{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "Member1": [
          {
            "userMemberId": "0ecb2fff-d5de-43d0-be60-f7f201c10d41",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Admin"
              }
            ]
```

```
          },
          {
            "userMemberId": "42b9d114-2c0c-4e96-a1ce-b645d5e92365",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Steward"
              }
            ]
          },
          {
            "groupMemberId": "5fc0cc5f-e30e-488c-94bc-acdea171219d",
            "User2": [
              {}
            ],
            "Role3": [
              {
                "roleName": "Admin"
              }
            ]
          }
        ],
        "Member4": [
          {
            "userMemberId": "0ecb2fff-d5de-43d0-be60-f7f201c10d41",
            "Group5": [
              {}
            ],
            "Role6": [
              {
                "groupRoleName": "Admin"
              }
            ]
          },
          {
            "userMemberId": "42b9d114-2c0c-4e96-a1ce-b645d5e92365",
            "Group5": [
              {}
            ],
            "Role6": [
              {
                "groupRoleName": "Steward"
              }
            ]
          },
          {
            "groupMemberId": "5fc0cc5f-e30e-488c-94bc-acdea171219d",
            "Group5": [
              {
                "groupId": "4eb1f4a9-14a3-4539-8afc-733925161179",
                "groupName": "admin"
              }
            ],
            "Role6": [
              {
                "groupRoleName": "Admin"
              }
            ]
          }
        ]
      }
    ]
  }
}
```

If you look at the different values of userMemberId and groupMemberId, you'll notice there are only 3 unique values in total.
When no further filtering is requested, adding twice the same entity effectively means selecting twice the same thing.

In the above case, we have 3 Member instances in total: two related to a User and one to a Group.

As a result, we receive one empty User for the Member linked to the Group and two empty Groups for each Member linked to a User.

Adding Filtering into the Mix

We'd need to discard the irrelevant Member results. Luckily, we got filtering available.

Here is how it goes:

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" },
        "Member": [
          {
            "Id": { "name": "userMemberId" },
            "User": {
              "Id": { "name": "userId" },
              "FirstName": { "name": "firstName" },
              "LastName": { "name": "lastName" }
            },
            "Role": {
              "Signifier": { "name": "userRoleName" }
            },
            "Filter": { "Field": { "name": "userId", "operator": "NOT_NULL" } }
          },
          {
            "Id": { "name": "groupMemberId" },
            "Group": {
              "Id": { "name": "groupId" },
              "GroupName": { "name": "groupName" }
            },
            "Role": {
              "Signifier": { "name": "groupRoleName" }
            },
            "Filter": { "Field": { "name": "groupId", "operator": "NOT_NULL" } }
          }
        ]
      }
    }
  }
}
```

"Filter" is a reserved key of the query language. In the above query, we first only keep the Members having a related User by specifying a "userId is not null" filtering clause. (More on available filters later in this guide)
Then, we select the related Members again, this time only keeping those having a related Group.

```
{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "Member1": [
          {
            "userMemberId": "0ecb2fff-d5de-43d0-be60-f7f201c10d41",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Admin"
              }
            ]
          },
          {
            "userMemberId": "42b9d114-2c0c-4e96-a1ce-b645d5e92365",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Steward"
              }
            ]
          }
        ],
        "Member4": [
          {
            "groupMemberId": "5fc0cc5f-e30e-488c-94bc-acdea171219d",
            "Group5": [
              {
                "groupId": "4eb1f4a9-14a3-4539-8afc-733925161179",
                "groupName": "admin"
              }
            ],
            "Role6": [
              {
                "groupRoleName": "Admin"
              }
            ]
          }
        ]
      }
    ]
  }
}
```


In the above result tree, "Member1" contains all related users and "Member4" only contains the groups. Just like we wanted.

Some Performance Considerations

Each time a "to-many" relation is traversed in the query tree, one could expect a performance hit as it means a new query is made against DGC's internal storage engine.
In the above example, the relation between the Community and Member entities is of the "to-many" kind as a Community can have *many* related Members.

Depending on the shape and amount of results, the performance penalty can range from completely irrelevant to a sizeable chunk added to the overall query time.

Here is thus the most performant way to query

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" },
        "Member": {
          "Id": { "name": "memberId" },
          "User": {
            "Id": { "name": "userId" },
            "FirstName": { "name": "firstName" },
            "LastName": { "name": "lastName" }
          },
          "Group": {
            "Id": { "name": "groupId" },
            "GroupName": { "name": "groupName" }
          },
          "Role": {
            "Signifier": { "name": "roleName" }
          }
        }
      }
    }
  }
}
```

For the sake of completeness, here is how the results would be formatted

```
{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "Member1": [
          {
            "memberId": "0ecb2fff-d5de-43d0-be60-f7f201c10d41",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Group3": [
              {}
            ],
            "Role4": [
              {
                "roleName": "Admin"
              }
            ]
          },
          {
            "memberId": "42b9d114-2c0c-4e96-a1ce-b645d5e92365",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Group3": [
              {}
            ],
            "Role4": [
              {
                "roleName": "Steward"
              }
            ]
          },
          {
            "memberId": "5fc0cc5f-e30e-488c-94bc-acdea171219d",
            "User2": [
              {}
            ],
            "Group3": [
              {
                "groupId": "4eb1f4a9-14a3-4539-8afc-733925161179",
                "groupName": "admin"
              }
            ],
            "Role4": [
              {
                "roleName": "Admin"
              }
            ]
          }
        ]
      }
    ]
  }
}
```

Sorting the Results

Ordering the results can be achieved using an "Order" clause. Just like for filters, "Order" references one or more declared fields on the entity to be sorted or one of its children, grand children, ...
Ordering can be requested in ascending or descending order using the "ASC" and "DESC" constants. Default is "ASC".

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" },
        "Order": [
          { "Field": { "name": "communityName", "order": "ASC" } }
        ]
      }
    }
  }
}
```

Suppose we'd like to order Business Terms using the name of a related entity, we'd build the query like so

```
{
  "ViewConfig": {
    "Resources": {
      "Term": {
        "Id": { "name": "id" },
        "Signifier": { "name": "name" },
        "Relation": {
          "type": "SOURCE",
          "Target": {
            "Id": { "name": "targetRelatedAssetId" },
            "Signifier": { "name": "targetRelatedAsset" }
          }
        },
        "Order": [
          { "Field": { "name": "targetRelatedAsset", "order": "ASC" } }
        ]
      }
    }
  }
}
```

The "type" property on the Relation allows determining which relationship is used when navigating from the parent Term to the Relation. More on this later in this document.

In the above query, there might be more than one targetRelatedAsset for each source Term. In that case, the query engine will first order the related target assets and use the first value found to order the parent Terms.

Just like filtering, the order clause only affects the entities on which it is set. In the above case, this means the targetRelatedAssets won't be sorted.
In order to have them also sorted, we need to add another ordering clause on the Relation entity.

Why not on the Target Term node, you might ask?
Ordering only makes sense when we have a collection.
Given that a Term may be source of many Relations and a Relation has only one Target Term, we need to sort the collection of Relations and not directly the related Target Term.

Here is the query that sorts both collections:
(Please note that there is no filtering in this query in order to keep it simple. Executing this would return all Assets and all relations available in Collibra. Depending on the system, this might be a lot...)

```
{
  "ViewConfig": {
    "Resources": {
      "Term": {
        "Id": { "name": "id" },
        "Signifier": { "name": "name" },
        "Relation": {
          "type": "SOURCE",
          "Target": {
            "Id": { "name": "targetRelatedAssetId" },
            "Signifier": { "name": "targetRelatedAsset" }
          },
          "Order": [
            { "Field": { "name": "targetRelatedAsset", "order": "ASC" } }
          ]
        },
        "Order": [
          { "Field": { "name": "targetRelatedAsset", "order": "ASC" } }
        ]
      }
    }
  }
}
```

Differentiating Selected Properties from Properties Required in a Filter Clause

Suppose we need to query the recently created Users. We can easily implement this with the "CreatedOn" property and a filter that uses the "greater than" operator.

Now, adding the "CreatedOn" property to the tree also selects that property, which might not be what we need (say we just want the user's id + first and last name)

This is how we can tell the query engine not to return the "CreatedOn" property and just use it in the filter:
("CreatedOn" is a date expressed as the number of milliseconds since 1/1/1970)

```
{
  "ViewConfig": {
    "Resources": {
      "User": {
        "Id": { "name": "userId" },
        "FirstName": { "name": "firstName" },
        "LastName": { "name": "lastName" },
        "CreatedOn": { "name": "createdOn", hidden: true },
        "Filter": { "Field": { "name": "createdOn", "operator": "GREATER", "value": "1440492290300" } }
      }
    }
  }
}
```

Using "hidden": true on a property removes that property from the results. The default value is false

```
{
    "view": {
        "User": [
            {
                "userId": "9546bbe9-7299-4a99-bfd2-d97f8256c201",
                "firstName": "Patrick",
                "lastName": "Star"
            },
            {
                "userId": "d9f3cc67-0db7-4aa5-a246-e83a62ea5c62",
                "firstName": "SpongeBob",
                "lastName": "SquarePants"
            }
        ]
    }
}
```

Stripping HTML from Text Results

DGC allows saving some values with HTML formatting tags in. These tags are not visible to the end user and are used by the user interface to know which pieces should be presented in bold or in a table, etc...

When querying such data, you might sometimes need to be able to strip the HTML formatting tags from the text value. E.g., because these tags appear like garbage in your Excel export.

```
    {
      "ViewConfig": {
        "Resources": {
          "Community": {
            "Id": { "name": "communityId" },
            "Name": { "name": "communityName" },
            "Description": { "name": "communityDescription", "stripHtml": true }
          }
        }
      }
    }
```

"stripHtml" can be used on any text field. When true, the returned value will be stripped from its HTML tags.

Filtering Operators

Operator	Reverse Operator	Parameters	Type compatibility	Description
EQUALS	NOT_EQUALS	1	Text, Number, Boolean	Equal / not equal to the given value
STARTS_WITH	NOT_STARTS_WITH	1	Text	The text starts / does not start with given characters
STARTS_WITH_DIGIT	/	Optional	Text	The text starts with a digit. The optional parameter is a pair of upper and lower boundaries separated by a comma. E.g., "3, 8" means any digit from 3 to 8 included.
ENDS_WITH	NOT_ENDS_WITH	1	Text	The text ends / does not end with given characters
INCLUDES	NOT_INCLUDES	1	Text	The text contains / does not contain the given characters
LESS	GREATER	1	Number	The value is strictly less than / greater than the given value
LESS_OR_EQUALS	GREATER_OR_EQUALS	1	Number	The value is less than or equal to / greater than or equal to the given value
BETWEEN	/	2	Number	The value is included within the given values
NULL	NOT_NULL	None	Text, Number, Boolean	Absence / presence of value

IN	NOT_IN	Collection	Text, Number, Boolean	The value is in / not in the set of given values
EXISTS	NOT_EXISTS	1 (optional)	n/a	See below
CR_FILTER_DOMAIN	/	1	n/a	ComplexRelation specific filter: include only ComplexRelations having at least one related asset in the given Domain

A sample for each operator

Operator	Example
EQUALS	`{ "Field": { "name": "domainName", "operator": "EQUALS", "value": "New Business Terms" } }`
STARTS_WITH	`{ "Field": { "name": "domainName", "operator": "STARTS_WITH", "value": "New" } }`
STARTS_WITH_DIGIT	`{ "Field": { "name": "assetName", "operator": "STARTS_WITH_DIGIT" } }`
ENDS_WITH	`{ "Field": { "name": "domainName", "operator": "ENDS_WITH", "value": "Terms" } }`
INCLUDES	`{ "Field": { "name": "domainName", "operator": "CONTAINS", "value": "Bus" } }`
LESS	`{ "Field": { "name": "lastModified", "operator": "GREATER", "value": "1440492290300" } }`
LESS_OR_EQUALS	`{ "Field": { "name": "lastModified", "operator": "GREATER_OR_EQUALS", "value": "1440492290300" } }`
BETWEEN	`{ "Field": { "name": "lastModified", "operator": "BETWEEN", "values": [ "1440492290300", "1440493000000" } }`
NULL	`{ "Field": { "name": "description", "operator": "NULL" } }`
IN	`{ "Field": { "name": "statusName", "operator": "IN", "values": [ "New", "In Review" ] } }`
EXISTS	`{ "Field": { "target": "RelationSource", "operator": "EXISTS", "value": "00000000-0000-0000-0000-000000007001", "name": "assetId" } }`
CR_FILTER_DOMAIN	`{ "Field": { "operator": "CR_FILTER_DOMAIN", "value": "00000000-0000-0000-0000-000000006013" } }`

EXISTS / NOT_EXISTS

This filter requires some additional explanations. EXISTS in the context of a graph query means testing the existence of a relationship with another entity. This is thus the only filter that is explicitly limited to filtering on an entity located directly under the filtered node.

In order to specify which relation should exist / not exist, the filter has a key named "target".

It is also possible to pass a parameter to the EXISTS filter. This parameter is used as a secondary filtering element.
Supposing you want to query the Assets having an Attribute of type "Description", you can use the EXISTS filter on the Asset with target value "Attribute" and also pass the Id of the "Description" type in the "value" key of the filter.

Please find below the possible values for "target" and the expected kind of value for the optional parameter:

Filtered Entity	Target value	Optional Parameter	Description
Community, Domain, Asset	Member	Role Id	Allows filtering Resources related / not related to a Member, optionally only Members related to the given Role Id
Asset	Relation	RelationType Id	Allows filtering Assets that are / are not "source *or* target" of a Relation, optionally only Relations related to the given RelationType Id
Asset	RelationSource	RelationType Id	Allows filtering Assets that are / are not "source " of a Relation, optionally only Relations related to the given RelationType Id
Asset	RelationTarget	RelationType Id	Allows filtering Assets that are / are not " target" of a Relation, optionally only Relations related to the given RelationType Id
Asset	Attribute	AttributeType Id	Allows filtering Assets that have / have not an Attribute, optionally only Attributes related to the given AttributeType Id

Asset	StringAttribute	AttributeType Id	Allows filtering Assets that have / have not a StringAttribute, optionally only StringAttributes related to the given AttributeType Id
Asset	SingleValueListAttribute	AttributeType Id	Allows filtering Assets that have / have not a SingleValueListAttribute, optionally only SingleValueListAttributes related to the given AttributeType Id
Asset	MultiValueListAttribute	AttributeType Id	Allows filtering Assets that have / have not a MultiValueListAttribute, optionally only MultiValueListAttribute related to the given AttributeType Id
Asset	BooleanAttribute	AttributeType Id	Allows filtering Assets that have / have not a BooleanAttribute, optionally only BooleanAttributes related to the given AttributeType Id
Asset	NumericAttribute	AttributeType Id	Allows filtering Assets that have / have not a NumericAttribute, optionally only NumericAttributes related to the given AttributeType Id
Asset	DateTimeAttribute	AttributeType Id	Allows filtering Assets that have / have not a DateTimeAttribute, optionally only DateTimeAttributes related to the given AttributeType Id

The EXISTS / NOT_EXISTS filters are exclusively for Communities, Domains and Assets

Filtering in Hierarchy

"EQUALS / NOT_EQUALS" and "IN / NOT_IN" operators, when used in conjunction with an Id property of an Asset, a RelationType or a Community, can take an additional "descendants": true parameter.
When true, the query engine will force an IN or NOT_IN filter and add all ids from the child Assets, RelationTypes or Communities to the given id or ids.

This allows selecting e.g.

- all Assets under a given Community, including the sub-Communities
- all Assets that are of type "X" or one of it's sub-types
- ...

Boolean Operators

The filtering operators can be combined using boolean operators. Combining boolean operators results in a logical binary tree of possibilities.

But such a binary tree is not easy to read, so the ViewConfig provides a handier way of specifying this: "Named Logical Array"

```
        "Filter": {
          "AND": [
            { "Field": { "name": "domainId", "operator": "EQUALS", "value": "02204077-1cd1-4c70-a7c4-
4cd845194b81" } },
            { "Field": { "name": "termrid", "operator": "EXISTS", "value": "00000000-0000-0000-0000-
000000007001", "target": "RelationSource" } },
            { "Field": { "name": "statusName", "operator": "IN", "values": [ "New", "In Review" ] } }
          ]
        }
```

Filtering elements that are bundled together in a named array, are logically combined using the name of the array: either "AND" or "OR".

These logical arrays can be nested as well, allowing all possible boolean combinations to take place

```
        "Filter": {
          "AND": [
            {
              "OR": [
                { "Field": { "name": "domainId", "operator": "EQUALS", "value": "02204077-1cd1-4c70-a7c4-
4cd845194b81" } },
                { "Field": { "name": "termrid", "operator": "EXISTS", "value": "00000000-0000-0000-0000-
000000007001", "target": "RelationSource" } }
              ]
            },
            { "Field": { "name": "statusName", "operator": "IN", "values": [ "New", "In Review" ] } }
          ]
        }
```

Filtering Properties

Because writing a JSON query for any non-trivial requirement is a tedious task, a few filter shortcuts have been added on some of the entities.
E.g., Relation has a "typeId" parameter which takes an id and alleviates the user from having to add a RelationType node with an id property just to enable filtering on a particular relation type.
These one-liner filtering properties are the most commonly used filters as they make the query a lot less verbose.

Here is an example on how to filter a StringAttribute on a given AttributeType using the "labelId" filtering property:

```
        "StringAttribute": {
            "labelId": "00000000-0000-0000-0000-000000000202",
            "Id": { "name": "descriptionId" },
            "LongExpression": { "name": "description" }
        }
```

The list of available filtering properties for each entity is available in the "Complete List of Entities, Properties and Relations" chapter below.

Virtual Properties

Some properties are not actually stored by the system but rather calculated at runtime. This means the value of the property is dynamically evaluated for each result when the query gets executed.
These virtual properties typically exist to support hierarchic queries where you want to know if the resource has children in the hierarchy or not (hasTaxonomyChildren, hasChildForRelation, ...)

Clarifying Relationship between two Entities

When 2 entities can be related to each other in more than one way, nesting the entities inside each others is not enough to determine which path to follow.
E.g.: an Asset can be either "source" or "target" of a Relation, a User might be the "creator" or the "lastModifier" of a Resource, ...

Depending on the entity at hand, there are essentially 2 possibilities:

- the name of the child entity is changed
  e.g. Source or Target should be used under Relation instead of Term
  -> these act and behave just like normal Terms and exist for the sole purpose of clarifying the relationship followed
- a special parameter called the "Parent Relationship Selector" is added to the child entity
  e.g. Relation has a "type" parameter with possible values "SOURCE" or "TARGET", this parameter determines the relationship between the Relation and the *parent* Term

So a query going two levels deep might look like this:

```
{
  "ViewConfig": {
    "Resources": {
      "Term": {
        "Id": { "name": "id" },
        "Signifier": { "name": "name" },
        "Relation": {
          "type": "SOURCE",
          "Target": {
            "Id": { "name": "relatedAssetLevelOneId" },
            "Signifier": { "name": "relatedAssetLevelOne" },
            "Relation": {
              "type": "TARGET",
              "Source": {
                "Id": { "name": "relatedAssetLevelTwoId" },
                "Signifier": { "name": "relatedAssetLevelTwo" }
              }
            }
          }
        }
      }
    }
  }
}
```

These special parameters and custom entity names only exists for a fraction of the available entities.

The question "What is available where?" is answered in the "Complete List of Entities, Properties and Relations" below.

Please note there is no filtering in the above query, executing this query means retrieving all assets and all their related assets 2 levels deep.
Probably huge on any production system, do not try at home.

Paging the Results

The Output Module also supports paging of the results for the root node of the query.

In other words, you can request to limit the results to a subset of the complete list by specifying an offset and a length parameter.

JSON key	Default value	Description
displayStart	0	The offset in the list of results. This offset is a zero based index value.
displayLength	-1	The maximum total number of results to return. A negative value means "unlimited"
maxCountLimit	-1	The maximum count value. A count of all records can lead to performance problems. When paging, it's possible to limit the max count to this value. Passing 0 means no count is done.

```
{
  "ViewConfig": {
    "displayStart": 10,
    "displayLength": 5,
            "maxCountLimit": 10000,
             "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "communityName" },
        "Description": { "name": "communityDescription" },
        "Order": [ { "Field": { "name": "communityName", "order": "ASC" } } ]
      }
    }
  }
}
```

The above query selects page 3 of all Communities using 5 results per page.

Please note that paged results should always be sorted, otherwise, the results might seem inconsistent from page to page.

Also take into consideration that paged results list will be recalculated on each request. All entities that have been added or removed in the meantime will show up / disappear from the list, thereby modifying the indexes of the elements in the result list.

DGC admin console allows limiting the number of results returned by queries. The values range from 10,000 to 100,000. If the option is enabled and the limit set then:

- default displayLength value (-1) will be overridden by the limit set through the console.
- If the displayLength set directly in the ViewConfig/TableViewConfig is larger than the limit value set in the console an exception will be thrown.

Mapping the Results to a Tabular Format

The Output Module supports different formats, some of them being tabular formats. Whenever you need a tabular output format, a different kind of ViewConfig should be used: the TableViewConfig.

TableViewConfig has a specific "Columns" mapping section allowing to assign each individual selected field to a particular column.

Up until now, we've been using the ViewConfig as input to the API which produces a JSON tree format. Let's now try using the TableViewConfig. This is conveniently available under the same **{{domain}}/rest/latest/outputModule/export/json** endpoint, just using the TableViewConfig as json payload.

```
    {
      "TableViewConfig": {
        "displayLength": 5,
        "displayStart": 10,
        "Resources": {
          "Community": {
            "Id": { "name": "communityId" },
            "Name": { "name": "communityName" },
            "Description": { "name": "communityDescription" }
          }
        },
        "Columns": [
          { "Column": { "fieldName": "communityId" } },
          { "Column": { "fieldName": "communityName" } },
          { "Column": { "fieldName": "communityDescription" } }
        ]
      }
    }
```

This query, when formatted, produces an array of rows, each containing the requested columns:

```
{
    "iTotalDisplayRecords": 48,
    "iTotalRecords": 5,
    "aaData": [
        {
            "communityId": "12345678-0006-0000-0000-000000000000",
            "communityName": "Simple Community 6",
            "communityDescription": ""
        },
        {
            "communityId": "12345678-0007-0000-0000-000000000000",
            "communityName": "Simple Community 7",
            "communityDescription": ""
        },
        {
            "communityId": "12345678-0008-0000-0000-000000000000",
            "communityName": "Simple Community 8",
            "communityDescription": ""
        },
        {
            "communityId": "12345678-0009-0000-0000-000000000000",
            "communityName": "Simple Community 9",
            "communityDescription": ""
        },
        {
            "communityId": "12345678-0010-0000-0000-000000000000",
            "communityName": "Simple Community 10",
            "communityDescription": ""
        }
    ]
}
```

Because, the "Columns" mapping determines what should be returned, setting "hidden": true on a property has no effect in a TableViewConfig.

JSON Data Table output contains the total number of available records in DCG for this query (iTotalDisplayRecords) as well as the number of records returned in this set (iTotalRecords)

It is thus possible to just get the count of entities without retrieving actual results by setting the "displayLength" value to 0:

```
    {
        "iTotalDisplayRecords": 48,
        "iTotalRecords": 0,
        "aaData": []
    }
```

TableViewConfig queries can be used with the following endpoints:

- {{domain}}/rest/latest/outputModule/export/{{json | csv}}
- {{domain}}/rest/latest/outputModule/export/{{json | csv | excel}}-file
- {{domain}}/rest/latest/outputModule/export/{{json | csv | excel}}-job

Handling "To-Many" Results in a Tabular Format

Suppose we want to select all Assets from a given Domain together with their "Note" attributes. Please note, each Asset may have multiple Notes. When there are multiple Notes, we'd like them to be ordered with the most recent Note at the top of the list.

The TableViewConfig could look something like this:

```
{
  "TableViewConfig": {
    "Resources": {
      "Term": {
        "Id": { "name": "termrId" },
        "Signifier": { "name": "termName" },
        "StringAttribute": {
          "LongExpression": { "name": "note" },
          "CreatedOn": { "name": "noteCreatedOn" },
          "Order": [ { "Field": { "name": "noteCreatedOn", "order": "DESC" } } ]
        },
        "Vocabulary": {
          "Id": { "name": "vocabularyId" }
        },
        "Filter": { "Field": { "name": "vocabularyId", "operator": "EQUALS", "value": "f342423f-54fd-4643-935b-
adbd9e7f5e25" } },
        "Order": [ { "Field": { "name": "termName" } } ]
      }
    },
    "Columns": [
      { "Column": { "fieldName": "termrId" } },
      { "Column": { "fieldName": "termName" } },
      { "Column": { "fieldName": "note" } }
    ]
  }
}
```

Depending on the format requested, the results might actually be quite different.

An Excel or CSV format would have each Asset duplicated on a new row for each Note value:

	A	B	C	D
1	termrId	termName	note	
2	c20d5b39-6c5d-411b-adcb-82a1dd3851cc	Business Term 1	Second Note	
3	c20d5b39-6c5d-411b-adcb-82a1dd3851cc	Business Term 1	First note	
4	1a6a8f73-43b0-4a29-84c3-baaa3467be70	Business Term 2	Single note on BT2	
5	7329349e-0631-41a7-a740-738979d887c6	Business Term 3	Single Note on BT3	
6				

If you're familiar with SQL queries, this is similar to joining two tables having a one-to-many relationship.
Unlike SQL though, supposing we were selecting a Term with two Notes and three Members, the Term would use three lines of the Excel table, not six, and the third row in the Note column would be empty.

Json format on the other hand won't add duplicate rows to the results and will instead return the first Note found, discarding other Notes altogether:

"First note" is missing for "Business Term 1"

```
{
    "iTotalDisplayRecords": 3,
    "iTotalRecords": 3,
    "aaData": [
        {
            "termrId": "c20d5b39-6c5d-411b-adcb-82a1dd3851cc",
            "termName": "Business Term 1",
            "note": "Second Note"
        },
        {
            "termrId": "1a6a8f73-43b0-4a29-84c3-baaa3467be70",
            "termName": "Business Term 2",
            "note": "Single note on BT2"
        },
        {
            "termrId": "7329349e-0631-41a7-a740-738979d887c6",
            "termName": "Business Term 3",
            "note": "Single Note on BT3"
        }
    ]
}
```

For tabular formats that don't duplicate rows, the "Group" mapping construct may be added to the "Columns" section:

```
{
  "TableViewConfig": {
    "Resources": {
      "Term": {
        "Id": { "name": "termrId" },
        "Signifier": { "name": "termName" },
        "StringAttribute": {
          "LongExpression": { "name": "note" },
          "CreatedOn": { "name": "noteCreatedOn" },
          "Order": [ { "Field": { "name": "noteCreatedOn", "order": "DESC" } } ]
        },
        "Vocabulary": {
          "Id": { "name": "vocabularyId" }
        },
        "Filter": { "Field": { "name": "vocabularyId", "operator": "EQUALS", "value": "f342423f-54fd-4643-935b-
adbd9e7f5e25" } },
        "Order": [ { "Field": { "name": "termName" } } ]
      }
    },
    "Columns": [
      { "Column": { "fieldName": "termrId" } },
      { "Column": { "fieldName": "termName" } },
      {
        "Group": {
          "name": "Notes",
          "Columns": [
            { "Column": { "fieldName": "note" } }
          ]
        }
      }
    ]
  }
}
```

A "Group" mapping allows grouping multiple results for a single parent. A "Group" must receive a user defined name that will be used when formatting the results.

```
{
    "iTotalDisplayRecords": 3,
    "iTotalRecords": 3,
    "aaData": [
        {
            "termrId": "c20d5b39-6c5d-411b-adcb-82a1dd3851cc",
            "termName": "Business Term 1",
            "Notes": [
                {
                    "note": "Second Note"
                },
                {
                    "note": "First note"
                }
            ]
        },
        {
            "termrId": "1a6a8f73-43b0-4a29-84c3-baaa3467be70",
            "termName": "Business Term 2",
            "Notes": [
                {
                    "note": "Single note on BT2"
                }
            ]
        },
        {
            "termrId": "7329349e-0631-41a7-a740-738979d887c6",
            "termName": "Business Term 3",
            "Notes": [
                {
                    "note": "Single Note on BT3"
                }
            ]
        }
    ]
}
```


Some rules about "Group"

- "Group" mappings cannot be nested, a "Group" defined within a "Group" is not supported
- All columns within a Group must be related to the same parent entity

Setting execution timeout

When query is very complicated or/and operates on big amount of data its execution may be slower than expected. In such case usually the best approach is to paginate the results as already described in other sections of this document. There are some cases where we're not yet aware of the complexity or amount of data we will be operating on, though. Timeout can be a good tool of breaking too long executions. Output Module has the ability to timeout not only on the execution logic level but also it can break running DB queries if they are taking too long and defend database load from unwanted stress.

Timeout can be set for each ViewConfig and TableViewConfig execution. It can be defined on main config level. Defining it in the body of the query is optional.

If timeout is not set in the ViewConfig (or TableViewConfig) then a default value (configurable from DGC console) will be added. DGC is shipped with this setting set to 8 hours. No single query may run longer than 24 hours and this is a max value that can be set. Pagination is recommended in case of queries that may run longer.

WARNING

- Those values will be made significantly smaller in the next major release so it would be prudent to already think about pagination.
- Queries if *queryTimeout* set to more then the limit value (24 hours) will have the *queryTimeout* overridden by the max limit value (24 hours).

Important exceptions are the **{{domain}}/rest/latest/outputModule/export/{{csv | excel}}-job** endpoints. Here data is calculated in chunks (with size of the chunk defined through the DGC console). Each chunk is calculated in a separate query and the timeout value set in the TableViewConfig will be a timeout value for calculation of a single chunk.

JSON key	Minimum value	Default value	Maximum value	Description
queryTimeout	1 minute	8 hours (configurable)	24 hours	Timeout in number of seconds that computation of the output can last. No decimal point allowed. Negative values are invalid. Zero means no timeout. Positive values will stop execution and return an error in case of execution takes longer than given number of seconds.

Example of ViewConfig with a timeout set:

```
{
   "ViewConfig": {
       "queryTimeout": 5,
       "Resources": {
           "Vocabulary": {
               "name": "d",
               "Name": {
                   "name": "vocName"
               },
               "Term": {
                   "name": "t",
                   "Signifier": {
                       "name": "termName"
                   },
                   "ConceptType": {
                       "name": "tt",
                       "Name": {
                           "name": "termType"
                       }
                   }
               }
           }
       }
   }
}
```

Once the timeout is reached the requestor of REST request will receive response with HTTP error code 408. Instead of results message body will contain a JSON with error description.

Structural Validation of the Query

Hand writing ViewConfigs and TableViewConfigs is a tedious and error prone task. All:

- {{domain}}/rest/latest/outputModule/export/{{xml | json | csv}}
- {{domain}}/rest/latest/outputModule/export/{{xml | json | csv | excel}}-file
- {{domain}}/rest/latest/outputModule/export/{{xml | json | csv | excel}}-job

endpoints allow using **validationEnabled** parameter. This parameter when set to true will enable validation of the input ViewConfig/TableViewConfig. By default the value of parameter is set to false.

Here is an example where there is a small typo in the filter: "userID" instead of "userId". If we do a POST request to:

{{domain}}/rest/latest/outputModule/export/json?validationEnabled=true

with the following body:

```
{
  "ViewConfig": {
    "displayLength": 5,
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "community" },
        "Member": {
          "Id":{ "name": "memberId"},
          "User": {
            "Id": { "name": "userId" },
            "FirstName": { "name": "userName" }
          }
        },
        "Filter": {"Field": {"name":"userID", "Operator":"NOT_NULL"}}
      }
    }
  }
}
```

The response will be:

```
{
    "viewConflict": [
        {
            "type": "View Configuration Conflict",
            "message": "Field 'userID' is unknown.",
            "id": "7c723d33-dc8d-484b-90df-91e3364d771a"
        }
    ]
}
```

Available API endpoints and query formats

Available rest api endpoints urls are:

- {{domain}}/rest/latest/outputModule/export/{{format}}
- {{domain}}/rest/latest/outputModule/export/{{format}}-file
- {{domain}}/rest/latest/outputModule/export/{{format}}-job

Available formats are: XML, JSON, CSV, EXCEL.

Endpoints vs formats

endpoint	CSV	JSON	CSV	EXCEL
• {{domain}}/rest/latest/outputModule/export/{{format}}	YES	YES	YES	NO
• {{domain}}/rest/latest/outputModule/export/{{format}}-file	YES	YES	YES	YES
• {{domain}}/rest/latest/outputModule/export/{{format}}-job	YES	YES	YES	YES

ViewConfig/TableViewConfig vs. format

format	supports ViewConfig	supports TableViewConfig

XML	YES	NO
JSON	YES	YES
CSV	NO	YES
EXCEL	NO	YES

Single query vs. multi query

Multi-query endpoints have less chance to timeout do to execution time limits and thus can be used for larger exports.

endpoint	CSV	JSON	CSV	EXCEL
• {{domain}}/rest/latest/outputModule/export/{{format}}	SINGLE	SINGLE	SINGLE	SINGLE
• {{domain}}/rest/latest/outputModule/export/{{format}}-file	SINGLE	SINGLE	SINGLE	SINGLE
• {{domain}}/rest/latest/outputModule/export/{{format}}-job	SINGLE	SINGLE	MULTI	MULTI

Complete List of Entities, Properties and Relations

Entity

Entity is the base abstract class of all other entities

An abstract entity cannot be queried, thus "Entity" cannot be used in the query tree.

Properties

id	Text (36)	Universally unique identifier (UUID)

Resource

extends Entity

Resource is an abstract entity which is the base class of most other entities.
Most other entities share thus the following properties and relations.

An abstract entity cannot be queried, thus "Resource" cannot be used in the query tree.

Properties

createdOn	Number	Creation date (# milliseconds since 1/1/1970)
createdBy	Text	Id of the user who created this Resource
lastModified	Number	Last modification date (# milliseconds since 1/1/1970)
lastModifiedBy	Text	Id of the last user who modified this Resource
system	Boolean	Is this resource reserved by the system

Relations

User	Many-to-one	• the User who created the Resource • the User who last modified the Resource • the User who created *or* last modified the resource (See User for details on specifying which kind of relationship is used)

Representation

extends Resource

Representation is an abstract entity which is the base class for Term.
All Terms share thus the following relationships.

An abstract entity cannot be queried, thus "Representation" cannot be used in the query tree.

Properties		
/		
Relations		
Status	Many-to-One	The current Status of this Representation
Vocabulary	Many-to-One	The Vocabulary containing this Representation
ConceptType	Many-to-One	The ConceptType of this Representation
Attribute	One-to-Many	The collection of Attributes present in this Representation
StringAttribute	One-to-Many	The collection of StringAttributes present in this Representation
ScriptAttribute	One-to-Many	The collection of ScriptAttributes present in this Representation
SingleValueListAttribute	One-to-Many	The collection of SingleValueListAttributes present in this Representation
MultiValueListAttribute	One-to-Many	The collection of MultiValueListAttributes present in this Representation
BooleanAttribute	One-to-Many	The collection of BooleanAttributes present in this Representation
NumericAttribute	One-to-Many	The collection of NumericAttributes present in this Representation
DateTimeAttribute	One-to-Many	The collection of DateTimeAttributes present in this Representation
DateAttribute	One-to-Many	The collection of DateAttributes present in this Representation

Community

extends Resource

Represents the hierarchy of Communities available in DGC.

Properties		
name	Text (255)	The name of the Community
description	Text	The description of the Community
uri	Text (255)	The uri of the Community
language	Text(255)	The name of the language used
meta	Boolean	Is the Community related to DGC's meta model (aka hidden Community)
hasNonMetaChildren	Boolean	Is the Community containing non meta sub-communities or domains (Vocabularies)
hasNonMetaChildCommunity	Boolean	Is the Community containing non meta communities
Relations		
ParentCommunity	Many-to-One	The parent Community of this Community (optional, null for root communities)
Community	One-to-Many	The collection of sub-Communities
Vocabulary	One-to-Many	The collection of Vocabularies contained in this Community
Member	One-to-Many	The collection of Members playing a Role in this Community
Filtering Property		
rootCommunity	Boolean	When true, the query engine will add a filter retaining only root Communities Only available when Community is also root of the query tree

ParentCommunity

extends Community

Exact synonym of Community.
Can only be used as a child of Community, in order to disambiguate the relationship followed.

Vocabulary

extends Resource

Represents a Vocabulary (aka Domain) available in one of DGC's Communities.

Properties		
name	Text (255)	The name of the Vocabulary
description	Text	The description of the Vocabulary
uri	Text (255)	The uri of the Vocabulary
meta	Boolean	Is the Vocabulary related to DGC's meta model (aka hidden Vocabulary)
Relations		
Term	One-to-Many	The collection of Terms (Assets) contained in this Vocabulary
Community	Many-to-One	The parent Community
VocabularyType	Many-to- One	The type of this Vocabulary
Member	One-to-Many	The collection of Members playing a Role in this Vocabulary
Mapping	One-to-Many	The collection of Mappings corresponding to this Vocabulary

VocabularyType

extends Resource

Each Vocabulary has a VocabularyType (aka DomainType).

Properties		
signifier	Text (255)	The name of the VocabularyType
name		synonym for signifier
description	Text	The description of the VocabularyType
meta	Boolean	Is the VocabularyType related to DGC's meta model
Relations		
Vocabulary	One-to-Many	The collection of Vocabulary instances having this VocabularyType
VocabularyType	Many-to-One	The parent VocabularyType of this VocabularyType (optional, null for root VocabularyTypes)
VocabularyTypeSpecializedConcepts	One-to-Many	The collection of VocabularyTypes children of this VocabularyType

VocabularyTypeSpecializedConcepts

extends VocabularyType
Collection of VocabularyType

Exact synonym of VocabularyType.
Can only be used as a child of VocabularyType, in order to disambiguate the relationship followed

BinaryFactType

extends Resource

A BinaryFactType (aka RelationType) defines a class of relationship between two ConceptTypes (AssetTypes).

Properties		
role	Text	The label of the relation when followed from head to tail
corole	Text	The label of the reversed relation, when followed from tail to head
description	Text	The description of the BinaryFactType
Relations		
Relation	One-to-Many	The collection of Relation instances having this BinaryFactType
HeadTerm	Many-to-One	The ConceptType that is head of this BinaryFactType HeadTerm is a synonym of Term and has the purpose of clarifying which path is followed from the Relation entity to its child. In this case the child node is the head.
TailTerm	Many-to-One	The ConceptType that is tail of this BinaryFactType TailTerm is a synonym of Term and has the purpose of clarifying which path is followed from the Relation entity to its child. In this case the child node is the tail.
Parent Relationship Selector		
type	This parameter allows specifying which path should be followed from the parent ConceptType entity to this BinaryFactType. The possible values are either "HEAD" or "TAIL" which tells whether the parent ConceptType is the head or the tail of the BinaryFactType. The default value is "HEAD".	

Relation

extends Resource

A Relation links two Terms (aka Assets) together

Properties		
startingDate	Number	The optional start date for this Relation
endingDate	Number	The optional end date for this Relation
isGenerated	Boolean	True if this Relation was generated
Relations		
BinaryFactType	Many-to-One	The type of this Relation
Source	Many-to-One	The source Term of this Relation
Target	Many-to-One	The target Term of this Relation
Parent Relationship Selector (only if parent is a Term node or is of type inheriting from Term node)		
type	This parameter allows specifying which path should be followed from the parent Term entity to this Relation. The possible values are either "SOURCE" or "TARGET" which tells whether the parent Term is the source or target of the Relation. This parameter is mandatory as there is no default value.	
Filtering Property		
typeId	Allows filtering Relations using the Id value of their related BinaryFactType	

ComplexRelation

extends Term

A ComplexRelation is an anonymous Term (Asset) whose signifier (name) has been generated.

Properties

/

Relations

ComplexRelationType	Many-to-One	The type of this complex relation

Filtering Property

typeId	Allows filtering ComplexRelations using the Id value of their related ComplexRelationType

Additional Parameters

separator	The character to be used to separate related Asset names in an Excel or CSV export
quote	The character to be used to quote related Asset names in an Excel or CSV export

ComplexRelationType

extends ConceptType

A ComplexRelationType determines the type of a ComplexRelation.

Properties

/

Relations

ComplexRelation	OneToMany	The collection of ComplexRelation instances having this ComplexRelationType
ComplexRelationLegType	OneToMany	The collection of ComplexRelationLegTypes linked to this ComplexRelationType
ComplexRelationAttributeType	OneToMany	The collection of ComplexRelationAttributeTypes linked to this ComplexRelationType

ComplexRelationLegType

extends Resource

A ComplexRelationLegType is a BinaryFactType used in the context of a ComplexRelationType.
The headTerm of those BinaryFactTypes being the ComplexRelationType.

Can only be used as a child of ComplexRelationType.

Properties

min	Number	The minimum occurrences of this BinaryFactType in the ComplexRelationType
max	Number	The maximum occurrences of this BinaryFactType in the ComplexRelationType
legOrder	Number	Order of this ComplexRelationLegType in the ComplexRelationType

Relations

BinaryFactType	Many-to-One	The BinaryFactType of this ComplexRelationLegType

ComplexRelationAttributeType

extends Resource

A ComplexRelationAttributeType is an AttributeType used in the context of a ComplexRelationType.

Can only be used as a child of ComplexRelationType.

Properties

min	Number	The minimum occurrences of this AttributeType in the ComplexRelaionType

max	Number	The maximum occurrences of this AttributeType in the ComplexRelaionType
readOnly	Boolean	Indicate if the attribute can be edited or not.
attributeOrder	Number	Order of this ComplexRelationAttributeType in the ComplexRelationType
Relations		
AttributeType	Many-to-One	The AttributeType of this ComplexRelationaAttributeType

Term

extends Representation

A Term (aka Asset) is the basic building block capturing informations about the assets available in DGC

Properties		
signifier	Text (2000)	The full name of the Term
displayName	Text (2000)	The display name of the Term
articulationScore	Number	Result of the last calculation of the articulation score

hasChildrenForRelation (**deprecated**)	Boolean	Virtual (calculated) property telling if this Term has children for the relation type defined at query level.
		This property takes 2 additional parameters: the BinaryFactType id and the direction (role or co-role)
		E.g.:
		`"HasChildrenForRelation": {`
		`  "name": "hasChildren",`
		`  "relationTypeId": "00000000-0000-0000-0000-000000007005",`
		`  "roleDirection": true`
		`}`
		
		Can only be used if Term is a root node of the query.
		Is not inherited by nodes extending Term node.

avgRating	Number	Average value of all ratings assigned to the Term
ratingsCount	Number	Number of all ratings signed to the Term
class	Text	Given other entities that extend Term, can be used to differentiate amongst the various sub-classes.
Relations		
Relation	One-to-Many	The collection of Relations this Term has (see Relation for a mandatory 'type' parameter)
Member	One-to-Many	The collection of Members this Term has
Mapping	One-to-Many	The related mappings
Tag	Many-to-Many	The collection of Tags associated to this Term
Filtering Property		

rootOfRelation	This filtering property is an array relation types / direction pairs. Root terms are not child of any of the given relations.
	E.g.:
	`"rootOfRelation": [`
	`  {`
	`  "relationTypeId": "00000000-0000-0000-0000-000000007038",`
	`  "roleDirection": true`
	`  },`
	`  {`
	`  "relationTypeId": "00000000-0000-0000-0000-000000007005",`
	`  "roleDirection": true`
	`  }`
	`],`	

Source

extends Term

Exact synonym of Term.
Can only be used as a child of Relation, in order to disambiguate the relationship followed.

Target

extends Term

Exact synonym of Term.
Can only be used as a child of Relation, in order to disambiguate the relationship followed.

HeadTerm

extends ConceptType

Exact synonym of ConceptType.
Can only be used as a child of BinaryFactType, in order to disambiguate the relationship followed.

TailTerm

extends ConceptType

Exact synonym of ConceptType.
Can only be used as a child of BinaryFactType, in order to disambiguate the relationship followed.

ConceptType

extends Resource

A ConceptType (aka AssetType) determines the type of Term (aka Asset)

Properties		
signifier	Text (255)	The name of this ConceptType
name		synonym for signifier
description	Text	The description of the ConceptType
meta	Boolean	Is the ConceptType related to DGC's meta model
color	Text	The color of the ConceptType
icon	Text	The icon of the ConceptType
acronym	Text	The acronym of the ConceptType
symbolType	Text	Defines it the icon or acronym should be used in DGC Possible values : ICON, ACRONYM, NONE
displayNameEnabled	Boolean	Is display name enabled for all Terms of this ConceptType
ratingEnabled	Boolean	Are ratings enabled for all Terms of this ConceptType
Relations		
Term	One-to-Many	The collection of Terms instances having this ConceptType
ConceptType	Many-to-One	The parent ConceptType of this ConceptType
ConceptTypeSpecializedConcepts	One-to-Many	The collection of ConceptTypes which have this ConceptType as parent

ConceptTypeSpecializedConcepts

extends ConceptType
Collection of ConceptType

Can only be used as a child of ConceptType, in order to disambiguate the relationship followed.
Please note that ComplexRelationType despite inheriting from ConceptType does not support ConceptTypeSpecializedConcepts node.

Attribute

extends Resource

Attribute represents an attribute linked to a Representation

Properties		
value	Text	The text value of this attribute
class	Text	Given other entities extend Attribute, you may use the "class" qualifier to differentiate amongst the various sub-classes
Relations		
AttributeType	Many-to-One	The type of attribute
Term	Many-to-One	The Term this Attribute belongs to
Filtering Property		
labelId	Allows filtering the Attributes based on the id of their related AttributeType	

StringAttribute

extends Attribute

A StringAttribute is an Attribute dedicated to text values

Properties		
longExpression	Text	The unbounded text value. It is right now obsolete and returning the same content as Attribute::value.

ScriptAttribute

extends Attribute

A ScriptAttribute is an Attribute dedicated to script values

Properties		
script	Text	The script. It is right now obsolete and returning the same content as Attribute::value.

SingleValueListAttribute

extends Attribute

A SingleValueListAttribute is an Atribute dedicated to storing a single value selected from a list

MultiValueListAttribute

extends Attribute

A MultiValueListAttribute is an Atribute dedicated to storing a multiple values selected from a list

Properties		
values	Text[]	The multiple values

BooleanAttribute

extends Attribute

A BooleanAttribute is an Attribute dedicated to boolean values

Properties		
booleanValue	Boolean	The value

NumericAttribute

extends Attribute

A NumericAttribute is an Attribute dedicated to numeric values

Properties		
numericValue	Number	The stored number

DateTimeAttribute

extends Attribute

A DateTimeAttribute is an Attribute dedicated to date values that also keep track of time

Properties		
dateTime	Number	The date and time values expressed as the number of milliseconds since 1/1/1970

DateAttribute

extends Attribute

A DateAttribute is an Attribute dedicated to date values

Properties		
date	Number	The date value expressed as the number of milliseconds since 1/1/1970

AttributeType

extends Resource

The AttributeType determines the type of an Attribute

Properties		
signifier	Text(255)	The name of this attributeType
name		synonym for signifier
description	Text	The description of this attributeType
attributeKind	Text(255)	The kind of the AttributeType, possible values are: BOOLEAN, STRING, NUMERIC,DATE, DATE_TIME, SINGLE_VALUE_LIST, MULTI_VALUE_LIST, SCRIPT
language	Text(255)	the name of the language used, in case kind is SCRIPT
isInteger	Boolean	indicate if AttributeType defines an integer (true) or decimal (false) number in case kind is NUMERIC
allowedValues	Text	comma separated list of values in case kind is SINGLE_VALUE_LIST or MULTI_VALUE_LIST
Relations		
Attribute	One-to-Many	The collection of Attributes of this type

User

extends Resource

User represents DGC's users.

Any Resource has a creation date and a last modification date. DGC also stores which User made each of these operations.
The User entity is thus Related to all types as the creator and/or last modifier of the entity.

Properties		
userName	Text	The user name (aka login)
firstName	Text	The first name
lastName	Text	The last name
fullName	Text	Virtual property containing the first and last name together (useful for contains filters)
gender	Text	The gender
language	Text	The user language
activated	Boolean	Is the user activated or not
ldapUser	Boolean	Is this an ldapUser
apiUser	Boolean	Is this an API user
enabled	Boolean	Is the user enabled
emailAddress	Text	The user's primary email address
guest	Boolean	Is this a guest user
Relations		
Email	Many-to-Many	The collection of Emails owned by this User
Phone	Many-to-Many	The collection of Phone numbers owned by this User
InstantMessagingAccount	Many-to-Many	The collection of InstantMessagingAccount accounts owned by this User
Website	Many-to-Many	The collection of Websites owned by this User
Address	Many-to-Many	The collection of Addresses owned by this User
Community	One-to-Many	The collection of Communities created or last modified by this User
Vocabulary	One-to-Many	The collection of Vocabularies created or last modified by this User
VocabularyType	One-to-Many	The collection of VocabularyTypes created or last modified by this User
BinaryFactType	One-to-Many	The collection of BinaryFactTypes created or last modified by this User
Relation	One-to-Many	The collection of Relations created or last modified by this User
ComplexRelation	One-to-Many	The collection of ComlexRelations created or last modified by this User
Term	One-to-Many	The collection of Terms created or last modified by this User
ConceptType	One-to-Many	The collection of ConceptTypes created or last modified by this User
Attribute	One-to-Many	The collection of Attributes created or last modified by this User
StringAttribute	One-to-Many	The collection of StringAttributes created or last modified by this User
ScriptAttribute	One-to-Many	The collection of ScriptAttributes created or last modified by this User
SingleValueListAttribute	One-to-Many	The collection of SingleValueListAttributes created or last modified by this User
MultiValueListAttribute	One-to-Many	The collection of MultiValueListAttributes created or last modified by this User
BooleanAttribute	One-to-Many	The collection of BooleanAttributes created or last modified by this User
NumericAttribute	One-to-Many	The collection of NumericAttributes created or last modified by this User
DateTimeAttribute	One-to-Many	The collection of DateTimeAttributes created or last modified by this User
DateAttribute	One-to-Many	The collection of DateAttributes created or last modified by this User

AttributeType	One-to-Many	The collection of AttributeTypes created or last modified by this User
User	One-to-Many	The collection of Users created or last modified by this User
Group	Many-to-Many	The collection of Groups this User belongs to
Member	One-to-Many	The collection of Member linking this User to a Role on a Term, Vocabulary or Community
Role	One-to-Many	The collection of Roles created or last modified by this User
Status	One-to-Many	The collection of Statuses created or last modified by this User
WorkflowTaskInfo (**deprecated**)	One-to-Many	The collection of WorkflowTaskInfos created or last modified by this User
Mapping	One-to-Many	The collection of Mappings created or last modified by this User
Parent Relationship Selector		
linkType	This parameter allows specifying which path should be followed from the parent Resource to this User. Please note that when the parent Resource is Member or Group, "linkType" is not used and the relationship defined for Member or Group is used. When User is the parent node, "linkType" determines the relationship with the child Resources that have a "created or last modified" kind of relationship (see above Relations). The possible values are "CREATED", "MODIFIED", "CREATED_OR_MODIFIED" or "CREATED OR MODIFIED". CREATED_OR_MODIFIED is the default value, but can only be used when User is root of the query tree. CREATED_OR_MODIFIED is turned into a simple CREATED when User is not root of the query.	

Email

extends Resource

Email represents one of the User's email addresses.
Can only be used as a child of User.

Properties

emailAddress	Text	The email address

Phone

extends Resource

Phone represents one of the User's phone numbers.
Can only be used as a child of User.

Properties

phoneNumber	Text	The phone number
phoneType	Text	The phone type (FAX, MOBILE, OTHER, PAGER, PRIVATE, WORK)

InstantMessagingAccount

extends Resource

InstantMessagingAccount represents one of the User's instant messaging account.
Can only be used as a child of User.

Properties

account	Text	The account id
type	Text	The IM type (AOL, GTALK, ICQ, JABBER, LIVE_MESSENGER, SKYPE, YAHOO_MESSENGER)

Website

extends Resource

Website represents one of the User's websites.
Can only be used as a child of User.

Properties

url	Text	The url of the website
type	Text	The type of website (FACEBOOK, LINKEDIN, MYSPACE, TWITTER, WEBSITE)

Address

extends Resource

Address represents one of the User's addresses.
Can only be used as a child of User.

Properties

street	Text	The street
number	Text	The street number
city	Text	The city
postalCode	Text	The zip code
state	Text	The state
country	Text	The country
type	Text	The address type (HOME, WORK)

Group

extends Resource

A Group is a named collection of Users

Properties

groupName	Text	The name of the group

Relations

User	One-to-Many	The Users the are part of this Group
Member	One-to-Many	The collection of Member linking this Group to a Role on a Term, Vocabulary or Community

Member

extends Resource

A Member links a User or Group with a Role on a given Term, Vocabulary or Community (mutually exclusive)

Properties

/

Relations

User	Many-to-One	The related User (empty if linked to a Group)
Group	Many-to-One	The related Group (empty if linked to a User)
Role	Many-to-One	The related role
Term	Many-to-One	The associated asset
Vocabulary	Many-to-One	The associated domain
Community	Many-to-One	The associated community

Filtering Property	
roleId	Allows filtering Member using the Id property of the related Role

Role

extends Term (deprecated)
will extend Resource in the next major release

The Role that a User plays (Steward, Admin, ...)

Status

extends Resource

The Status of a Term

Properties		
signifier	Text(255)	The name of the status
description	Text	The status description
Relations		
Term	One-to-Many	The Terms having this Status

WorkflowTaskInfo (deprecated)

extends Resource

WorkflowTaskInfo holds all informations about an ongoing workflow task

Properties		
description	Text	The description of the task
title	Text	The title of the task
dueDate	Number	The due date of the task expressed as the number of milliseconds since 1/1/1970
itemResourceId	Text	The related item id
itemResourceType	Text	The related resource type
itemVerbalised	Text	The verbalised version of the related item
taskType	Text	The type of task
assignee	Text	The id of the assigned user
candidateUsers	Text	The ids or candidate users
vocabulary	Text	The related Vocabulary id
community	Text	The related Community id
status	Text	The status of the task

Mapping

extends Resource

A Mapping links an externally defined entity (Term or Vocabulary) to one of DGC's entities

Properties		
extSystemId	Text	The identifier of the external system
extEntityId	Text	The external identifier of the entity

extEntityUrl	Text	The external url of the entity
lastSyncDate	Number	The last synchronisation date
syncAction	Text	The last synchronization action (ADD, UPDATE or REMOVE)
description	Text	Description of this mapping
Relations		
Term	Many-to-One	The related Term
Vocabulary	Many-to-One	The related Vocabulary

Tag

extends Resource

A Tag allows categorising Assets by adding one or more labels to them

Properties		
name	Text	The name of the Tag
Relations		
Term	Many-to-Many	The related Terms

DataQualityRule (depracated)

extends Resource

A DataQualityRule describes the rules for the data quality of an asset

Properties		
name	Text	The name of the DataQualityRule
description	Text	The description of the DataQualityRule
Relations		

Scope

extends Resource

A Scope describes the scope of an assignment

Properties		
name	Text	The name of the scope
description	Text	The description of the scope
Relations		

DataType (deprecated)

extends Entity

A DataType is a Catalog entity that characterises a Data Element's data type

Properties		
name	Text	The name of the type (e.g. "Date" or "SSN")
description	Text	Description of the type
class	Text	There are 2 classes of DataTypes: BASE and ADVANCED

logicalDataType	Text	The corresponding logical data type as used by the profiling job (one of the base types)
Relations		
DataTypeMatch	One-to-Many	The related DataTypeMatches holding a specific % of match value for a given Data Element instance

AdvancedDataType (deprecated)

extends DataType

An AdvancedDataType is an extension of one of the base DataTypes (Text, Numeric, Date, ...) that provides patterns helping the profiling job detect the Data Type

Properties		
Relations		
DataTypePattern	One-to-Many	The patterns associated with this advanced data type

DataTypePattern (deprecated)

extends Entity

A DataTypePattern contains a pattern associated with an AdvancedDataType

Properties		
value	Text	The pattern
Relations		
AdvancedDataType	Many-to-One	The related AdvancedDataType

DataTypeMatch (deprecated)

extends Entity

A DataTypeMatch contains profiling results indicating what percentage of the actual data behind a DataElement asset (e.g. a database Column) matches a given DataType

Properties		
percentage	Double	The matching percentage
Relations		
Term	Many-to-One	The related Data Element
DataType	Many-to-One	The matched DataType

BaseView (deprecated)

extends Resource

An abstract entity base class of View and DiagramPicture

Properties		
name	Text	The name of the baseView
description	Text	The description of the baseView
config	Text	The (json) config of the baseView
originalView	Text	Id of the originalView of this base view, meaning the view this base view was created from
isDefault	Boolean	Is this a default baseView
isPreferred	Boolean	Is this a preferred (pinned) baseView

Relations

View (deprecated)

extends BaseView

A DGC View
Properties
Relations

DiagramPicture (deprecated)

extends BaseView

A DGC Diagram Picture		
Properties		
svg	Text	Text field containing an svg representation of the diagram picture
Relations		
View	Many-to-One	The view used to create / take the picture
DiagramPictureSharingRule	One-to-Many	The sharing rules of the diagram picture
AssignmentRule	Many-to-Many	The assignment rules of the diagram picture

DiagramPictureSharingRule (deprecated)

extends Resource

A DGC Diagram Picture Sharing Rule, a diagram Picture can be shared to a user, group, role		
Properties		
Relations		
Role	Many-to-One	The role linked to this rule
Group	Many-to-One	The group linked to this rule
User	Many-to-One	The user linked to this rule

AssignmentRule (deprecated)

extends Resource

An assignment rule, currently only exposed to graph query engine to show the Asset linked to a DiagramPicture		
Properties		
Relations		
Term	Many-to-One	The asset linked to this rule

Changes from the previous version

- References to the deprecated REST API v1 were removed from the document.
- Timeout mechanism described.
- Result limit mechanism described.
- API endpoints described.