

Output Module

Hitchhiker's Guide

The Hitchhiker's Guide to the Output Module

Revision: 08 Sep 2022

You can find the most up-to-date technical documentation on our Developer portal at

<https://developer.collibra.com/rest/output-module/>

© 2022 Collibra. All Rights Reserved.

Contents

| | |
|---|----|
| Contents | ii |
| What's new | 1 |
| Introduction | 2 |
| Prerequisites | 3 |
| Terminology | 3 |
| The Output Module query language | 5 |
| Getting started | 7 |
| Add related entities to the tree | 10 |
| Specify an entity alias | 13 |
| Add a related entity more than once | 15 |
| Add filtering | 20 |
| Sort the results | 28 |
| Differentiate selected properties from properties required in a filter clause | 32 |
| Strip HTML from text results | 34 |
| Filtering operators | 35 |
| Boolean operators | 42 |
| Filter properties | 45 |
| Virtual properties | 45 |
| Clarify the relationship between two entities | 46 |
| Page the results | 48 |
| Map the results to a tabular format | 50 |
| Set an execution timeout | 60 |
| Structural validation of the query | 63 |

| | |
|--|----|
| API endpoints and query formats | 66 |
| Endpoints and formats | 67 |
| ViewConfig/TableViewConfig and formats | 67 |
| Single query and multi-query | 67 |
| Entities, properties and relations | 69 |
| Entity | 69 |
| Resource | 69 |
| Representation | 70 |
| Organization | 72 |
| Community | 74 |
| ParentCommunity | 74 |
| Domain | 74 |
| DomainType | 75 |
| ChildDomainTypes | 75 |
| RelationType | 76 |
| Relation | 77 |
| ComplexRelation | 78 |
| ComplexRelationType | 79 |
| ComplexRelationLegType | 80 |
| ComplexRelationAttributeType | 80 |
| Asset | 81 |
| SourceAsset | 83 |
| TargetAsset | 84 |
| SourceAssetType | 84 |
| TargetAssetType | 84 |
| AssetType | 84 |

| | |
|-------------------------------------|-----|
| ChildAssetTypes | 86 |
| Attribute | 86 |
| StringAttribute | 87 |
| ScriptAttribute | 87 |
| SingleValueListAttribute | 87 |
| MultiValueListAttribute | 88 |
| BooleanAttribute | 88 |
| NumericAttribute | 88 |
| DateTimeAttribute | 89 |
| DateAttribute | 89 |
| AttributeType | 89 |
| User | 90 |
| Email | 95 |
| Phone | 95 |
| InstantMessagingAccount | 96 |
| Website | 96 |
| Address | 97 |
| Group | 97 |
| Responsibility | 98 |
| Role | 99 |
| Status | 99 |
| WorkflowTaskInfo (deprecated) | 99 |
| Mapping | 100 |
| Tag | 101 |
| DataQualityRule (deprecated) | 101 |
| Scope | 102 |

| | |
|--|-----|
| Comment | 102 |
| ParentComment | 103 |
| DataType (deprecated) | 103 |
| AdvancedDataType (deprecated) | 104 |
| DataTypePattern (deprecated) | 105 |
| DataTypeMatch (deprecated) | 105 |
| BaseView (deprecated) | 106 |
| View (deprecated) | 106 |
| DiagramPicture (deprecated) | 107 |
| DiagramPictureSharingRule (deprecated) | 107 |
| AssignmentRule (deprecated) | 108 |

What's new

- The Community and Domain entities are now extensions of Organization. (January 2022)
- The Output Module API uses the same terminology as the user interface. (September 2021)
- The guide now contains YAML examples.
- References to the deprecated REST API v1 were removed.
- The Timeout mechanism is described.
- The Result limit mechanism is described.
- The API endpoints are described.

Introduction

The Output Module is a lightweight graph query engine exposed through the public API. It allows different output formats, such as JSON, XML, Excel, and CSV. It also provides a single API to query most of the Collibra entities, such as assets, communities, domains and types, using SQL-like filtering capabilities. You can sort entities using any of the available properties and page results and view permissions for authenticated users who issue REST calls.



Prerequisites

Before you begin using the query language used in the Output Module, you must understand the Collibra API model and how to execute REST calls. This guide shows examples that query the REST API but does not explain how to execute REST calls. Refer to external online resources for tutorials and instructional resources.

Terminology

The Collibra API model was based on the Semantics of Business Vocabulary and Rules (SBVR) standard. Over time, the user interface adopted a simpler terminology set that aligns with Collibra concepts. Since version 2021.09 (5.7.10 for on-premisses), the Output Module API uses the same terminology as the user interface while the legacy one is deprecated.

The following table lists the renamed terminology:

| Deprecated | Current |
|-----------------------------------|------------------|
| Term | Asset |
| ConceptType | AssetType |
| ConceptTypeSpecializedConcepts | ChildAssetTypes |
| Vocabulary | Domain |
| VocabularyType | DomainType |
| VocabularyTypeSpecializedConcepts | ChildDomainTypes |
| Source | SourceAsset |

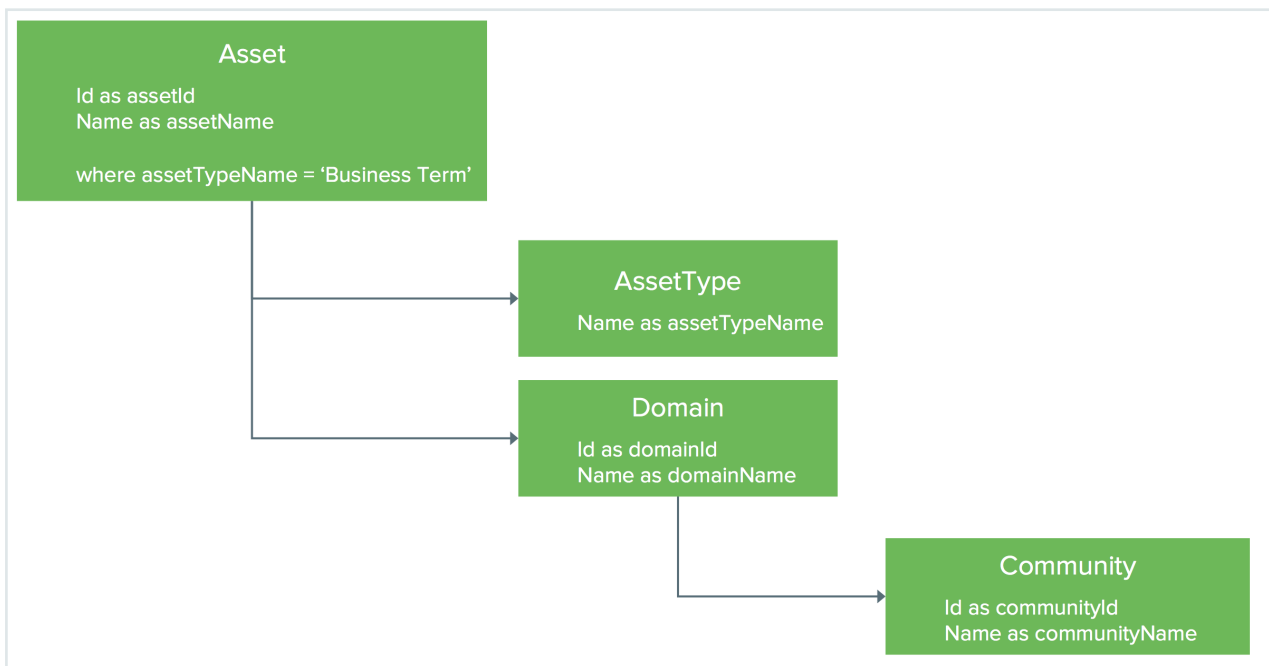
| Deprecated | Current |
|----------------|-----------------|
| Target | TargetAsset |
| BinaryFactType | RelationType |
| HeadTerm | SourceAssetType |
| TailTerm | TargetAssetType |
| Member | Responsibility |

Tip Use only the new terminology.

The Output Module query language

The API model has a set of well-defined entities and relations that allow you to create a single-rooted tree graph query and specify constraints that must exist for any of the resulting nodes, such as results filtering.

For example, to query all assets of type **Business Term** and their respective domain and community, specify the following tree graph:



Note

- The graph is a single-rooted tree graph.
- Multiple root nodes are not allowed.
- Each node has one parent.
- For each of the selected properties, you must specify a unique alias within the graph query.
- Filtering is specified on the node you want to filter and can reference any property of the current node or children or grandchildren. The example above shows assets filtered by their related AssetType name.

In this chapter

| | |
|---|----|
| Getting started | 7 |
| Add related entities to the tree | 10 |
| Specify an entity alias | 13 |
| Add a related entity more than once | 15 |
| Add filtering | 20 |
| Sort the results | 28 |
| Differentiate selected properties from properties required in a filter clause | 32 |
| Strip HTML from text results | 34 |
| Filtering operators | 35 |
| Boolean operators | 42 |
| Filter properties | 45 |
| Virtual properties | 45 |
| Clarify the relationship between two entities | 46 |
| Page the results | 48 |
| Map the results to a tabular format | 50 |
| Set an execution timeout | 60 |
| Structural validation of the query | 63 |

Getting started

The format of the query language is either JSON or YAML. For simplicity, this example starts with a basic query and builds from there.

Select the `Id` and `Name` for all communities as a flat list. The object representing the query is called `ViewConfig`, as it defines a particular view, which is a selection of the data. The object containing the graph part of the query is called `Resources`.

The following example shows the `Community` entity along with its `Id` and `Name` properties.

JSON

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "name": "Communities",
        "Id": { "name": "community id" },
        "Name": { "name": "community name" }
      }
    }
  }
}
```

<---\ ---- a node can (or must) have a name. Thus the community own 'name' property must be uppercased to avoid conflicts. <---/

YAML

```
---
ViewConfig:
  Resources:
    Community:
      name: "Communities"
      Id:
      Name:
      name: "community id"
      name: "community name"
```

<---\ ---- a node can (or must) have a name. Thus the community own 'name' property must be uppercased to avoid <---/

Note

- Entity and property keys are case insensitive, so `Community` and `Id` can be written in any case.
- The other keys are case sensitive. For example, `ViewConfig`, `Resources` or `Name` must be written as shown.
- If a property is spelled out the same way as a reserved keyword, you must use a different casing than the reserved key. For example, you use lowercase `name` as the node name and capitalized `Name` as the community name.

Test the API

To test the API, use a REST client, such as the Postman plugin for Chrome. Many output formats are available, but the JSON tree is the format that most resembles the query.

This example uses the following endpoint on the `OutputView` resource:

- `{{domain}}/rest/2.0/outputModule/export/json`

Use a POST call with the following body.

JSON

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": {
          "name": "community id"
        },
        "Name": {
          "name": "community name"
        }
      }
    }
  }
}
```

YAML

```
---
ViewConfig:
  Resources:
    Community:
      Id:
        name: "community id"
      Name:
        name: "community name"
```

Note Remember to set the content type header.

JSON

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

YAML

```
'Content-Type': 'application/x-yaml'
```

The output is 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"
      }
    ]
  }
}
```

You can use the ViewConfig queries with the following endpoints:

- `{{domain}}/rest/2.0/outputModule/export/{{xml | json}}`
- `{{domain}}/rest/2.0/outputModule/export/{{xml | json}}-file`
- `{{domain}}/rest/2.0/outputModule/export/{{xml | json}}-job`

Add related entities to the tree

Use this query example to add the users that have been assigned a role at the community level. To reach those entities, you must retrieve the `Responsibility` entities that represent the assignments between a user, a role and one of the following resources:

- **Asset**
- **Domain**
- **Community**

JSON

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "Id": { "name": "community id" },
        "Name": { "name": "community name" },
        "Responsibility": {
          "User": {
            "Id": { "name": "user id" },
            "FirstName": { "name": "first name" },
            "LastName": { "name": "last name" }
          },
          "Role": {
            "Signifier": { "name": "role name" }
          }
        }
      }
    }
  }
}
```


YAML

```

---
ViewConfig:
  Resources:
    Community:
      Id:
        name: "community id"
      Name:
        name: "community name"
      Responsibility:
        User:
          Id:
            name: "user id"
          FirstName:
            name: "first name"
          LastName:
            name: "last name"
        Role:
          Signifier:
            name: "role name"

```

Navigating from one entity to another requires nesting the entities. For a complete list of properties and relations for each entity, see [Entities, properties and relations](#).

The following is an example of how the results is formatted.

```

{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "Responsibility1": [
          {
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-00000000900002",
                "firstName": "Admin",

```

```

        "lastName": "Istrator"
      },
      "Role3": [
        {
          "roleName": "Admin"
        }
      ]
    },
    {
      "User2": [
        {
          "userId": "000000000-0000-0000-0000-0000000900002",
          "firstName": "Admin",
          "lastName": "Istrator"
        }
      ],
      "Role3": [
        {
          "roleName": "Steward"
        }
      ]
    }
  ]
}

```

Note

- The `ViewConfig` result tree always uses arrays for related entities, even when relations have a max cardinality of 1.
- Each responsibility has a maximum of one user and one role, even when arrays return.
- The results tree uses a generated entity alias in the response. For example, `Community0`, `Responsibility1` or `User2`.
- To prevent duplicate names in the JSON keys, an index number is concatenated to the entity name.
- The relationship from community to responsibility is optional. The query engine recognizes optional and required relations between entities, which is why `First Community` appears even when no users have roles.

Specify an entity alias

Auto-generated aliases in the response are not straightforward. For example, Community0, Responsibility1 or User2. For this reason, you must specify an alias.

JSON

```
{
  "ViewConfig": {
    "Resources": {
      "Community": {
        "name": "community",
        "Id": { "name": "community id" },
        "Name": { "name": "community name" },
        "Responsibility": {
          "name": "responsibility",
          "User": {
            "name": "employee",
            "Id": { "name": "user id" },
            "FirstName": { "name": "first name" },
            "LastName": { "name": "last name" }
          },
          "Role": {
            "name": "role",
            "Signifier": { "name": "role name" }
          }
        }
      }
    }
  }
}
```

YAML

```

---
ViewConfig:
  Resources:
    Community:
      name: "community"
      Id:
        name: "community id"
      Name:
        name: "community name"
      Responsibility:
        name: "responsibility"
      User:
        name: "employee"
        Id:
          name: "user id"
        FirstName:
          name: "first name"
        LastName:
          name: "last name"
      Role:
        name: "role"
      Signifier:
        name: "role name"

```

The results should then parse like the example below.

```

{
  "view": {
    "community": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "responsibility": [
          {
            "employee": [
              {
                "userId": "00000000-0000-0000-0000-0000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ]
          }
        ]
      }
    ]
  }
}

```

```

    }
  ],
  "role": [
    {
      "roleName": "Admin"
    }
  ]
},
{
  "employee": [
    {
      "userId": "000000000-0000-0000-0000-0000000900002",
      "firstName": "Admin",
      "lastName": "Istrator"
    }
  ],
  "role": [
    {
      "roleName": "Steward"
    }
  ]
}
]
}
]
}
}

```

Add a related entity more than once

To understand what roles users have in communities, you must query the groups that are linked through a responsibility.

To add another relation from community to responsibility, select the related groups.

This example shows the `Id` property of the two-responsibility nodes selected.

JSON

```

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

```

YAML

```

---
ViewConfig:
  Resources:
    Community:
      Id:
        name: "communityId"
      Name:
        name: "communityName"
      Responsibility:
        - Id:
            name: "userResponsibilityId"
          User:
            Id:
              name: "userId"
            FirstName:
              name: "firstName"
            LastName:
              name: "lastName"
            Role:
              Signifier:
                name: "userRoleName"
        - Id:
            name: "groupResponsibilityId"
          Group:
            Id:
              name: "groupId"
            GroupName:
              name: "groupName"
            Role:
              Signifier:
                name: "groupRoleName"

```

To add the same related entity twice under the same node, change the JSON object into an array. In this case, the `Responsibility` JSON object became an array, and the anonymous JSON objects composing the array are multiple responsibilities.

If you add the admin group to the second community, the results would be formatted similar to the example below.

```

{
  "view": {
    "Community0": [

```

```

    {
      "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
      "communityName": "First Community"
    },
    {
      "communityId": "12345678-0020-0000-0000-000000000000",
      "communityName": "Second Community",
      "Responsibility1": [
        {
          "userResponsibilityId": "0ecb2fff-d5de-43d0-be60-
f7f201c10d41",
          "User2": [
            {
              "userId": "00000000-0000-0000-0000-
000000900002",
              "firstName": "Admin",
              "lastName": "Istrator"
            }
          ],
          "Role3": [
            {
              "roleName": "Admin"
            }
          ]
        },
        {
          "userResponsibilityId": "42b9d114-2c0c-4e96-a1ce-
b645d5e92365",
          "User2": [
            {
              "userId": "00000000-0000-0000-0000-
000000900002",
              "firstName": "Admin",
              "lastName": "Istrator"
            }
          ],
          "Role3": [
            {
              "roleName": "Steward"
            }
          ]
        },
        {
          "groupResponsibilityId": "5fc0cc5f-e30e-488c-94bc-
acdea171219d",
          "User2": [
            {}
          ],
          "Role3": [
            {

```



```

        "roleName": "Admin"
      }
    ]
  },
  "Responsibility4": [
    {
      "userResponsibilityId": "0ecb2fff-d5de-43d0-be60-
f7f201c10d41",
      "Group5": [
        {}
      ],
      "Role6": [
        {
          "groupRoleName": "Admin"
        }
      ]
    },
    {
      "userResponsibilityId": "42b9d114-2c0c-4e96-a1ce-
b645d5e92365",
      "Group5": [
        {}
      ],
      "Role6": [
        {
          "groupRoleName": "Steward"
        }
      ]
    },
    {
      "groupResponsibilityId": "5fc0cc5f-e30e-488c-94bc-
acdea171219d",
      "Group5": [
        {
          "groupId": "4eb1f4a9-14a3-4539-8afc-
733925161179",
          "groupName": "admin"
        }
      ],
      "Role6": [
        {
          "groupRoleName": "Admin"
        }
      ]
    }
  ]
}

```

```
}
```

Note In the example above, the `userResponsibilityId` and `groupResponsibilityId` values contain three unique values in total: two related to a user and one to a group. When no further filtering is requested, adding the same entity twice means selecting the same thing twice. The result is one empty user for the responsibility linked to the group and two empty groups for each responsibility linked to a user.

Add filtering

To discard irrelevant responsibility results, use filtering.

JSON

```

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

```

YAML

```

---
ViewConfig:
  Resources:
    Community:
      Id:
        name: "communityId"
      Name:
        name: "communityName"
      Responsibility:
        -
          Id:
            name: "userResponsibilityId"
          User:
            Id:
              name: "userId"
            FirstName:
              name: "firstName"
            LastName:
              name: "lastName"
          Role:
            Signifier:
              name: "userRoleName"
          Filter:
            Field:
              name: "userId"
              operator: "NOT_NULL"
        -
          Id:
            name: "groupResponsibilityId"
          Group:
            Id:
              name: "groupId"
            GroupName:
              name: "groupName"
          Role:
            Signifier:
              name: "groupRoleName"
          Filter:
            Field:
              name: "groupId"
              operator: "NOT_NULL"

```

`Filter` is a reserved key. The example above first includes a `userId is not null` filtering clause to show responsibilities with a related user by (More on available filters later in this

guide). Then, select the related responsibilities again, this time only keeping those with a related group.

```
{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "Responsibility1": [
          {
            "userResponsibilityId": "0ecb2fff-d5de-43d0-be60-f7f201c10d41",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Admin"
              }
            ]
          },
          {
            "userResponsibilityId": "42b9d114-2c0c-4e96-a1ce-b645d5e92365",
            "User2": [
              {
                "userId": "00000000-0000-0000-0000-000000900002",
                "firstName": "Admin",
                "lastName": "Istrator"
              }
            ],
            "Role3": [
              {
                "roleName": "Steward"
              }
            ]
          }
        ],
        "Responsibility4": [
```

```

    {
      "groupResponsibilityId": "5fc0cc5f-e30e-488c-94bc-
acdea171219d",
      "Group5": [
        {
          "groupId": "4eb1f4a9-14a3-4539-8afc-
733925161179",
          "groupName": "admin"
        }
      ],
      "Role6": [
        {
          "groupRoleName": "Admin"
        }
      ]
    }
  ]
}

```

Note In the result tree, `Responsibility1` shows all related users and `Responsibility4` only contains the groups.

Filtering performance considerations

When a to-many relation is traversed in the query tree, performance is impacted because a new query is made against the Collibra internal storage engine. In the above example, the relation between the community and responsibility entities is of the to-many kind because a community can have many related responsibilities. 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 the optimal way to query.

JSON

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

YAML

```

---
ViewConfig:
  Resources:
    Community:
      Id:
        name: "communityId"
      Name:
        name: "communityName"
      Responsibility:
      Id:
        name: "ResponsibilityId"
      User:
      Id:
        name: "userId"
      FirstName:
        name: "firstName"
      LastName:
        name: "lastName"
      Group:
      Id:
        name: "groupId"
      GroupName:
        name: "groupName"
      Role:
      Signifier:
        name: "roleName"

```

The results should be formatted like the example below.

```

{
  "view": {
    "Community0": [
      {
        "communityId": "c87f166e-041f-4bea-8ff7-c1ffbab2ceeb",
        "communityName": "First Community"
      },
      {
        "communityId": "12345678-0020-0000-0000-000000000000",
        "communityName": "Second Community",
        "Responsibility1": [
          {
            "responsibilityId": "0ecb2fff-d5de-43d0-be60-f7f201c10d41",
            "User2": [

```



```

        {
            "userId": "000000000-0000-0000-0000-0000000900002",
            "firstName": "Admin",
            "lastName": "Istrator"
        }
    ],
    "Group3": [
        {}
    ],
    "Role4": [
        {
            "roleName": "Admin"
        }
    ]
},
{
    "responsibilityId": "42b9d114-2c0c-4e96-a1ce-b645d5e92365",
    "User2": [
        {
            "userId": "000000000-0000-0000-0000-0000000900002",
            "firstName": "Admin",
            "lastName": "Istrator"
        }
    ],
    "Group3": [
        {}
    ],
    "Role4": [
        {
            "roleName": "Steward"
        }
    ]
},
{
    "responsibilityId": "5fc0cc5f-e30e-488c-94bc-acdea171219d",
    "User2": [
        {}
    ],
    "Group3": [
        {
            "groupId": "4eb1f4a9-14a3-4539-8afc-733925161179",
            "groupName": "admin"
        }
    ],
    "Role4": [

```

```

        {
          "roleName": "Admin"
        }
      ]
    }
  ]
}

```

Sort the results

Use the `Order` clause to sort results. Just like filters, `Order` references one or more declared fields on the entity to be sorted or one of its children, or grandchildren.

Use the `ASC`, which is the default, and `DESC` constants to request ordering in ascending or descending order.

JSON

```

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

```

YAML

```

---
ViewConfig:
  Resources:
    Community:
      Id:
        name: "communityId"
      Name:
        name: "communityName"
      Order:
        -
          Field:
            name: "communityName"
            order: "ASC"

```

The following example shows assets ordered by the name of a related entity.

JSON

```

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

```

YAML

```

---
ViewConfig:
  Resources:
    Asset:
      Id:
        name: "id"
      Signifier:
        name: "name"
      Relation:
        type: "SOURCE"
      TargetAsset:
        Id:
          name: "targetRelatedAssetId"
        Signifier:
          name: "targetRelatedAsset"
    Order:
      -
        Field:
          name: "targetRelatedAsset"
          order: "ASC"

```

The `type` property on the relation allows you to determine which relationship is used when navigating from the parent asset to the relation. In the example above, there might be more than one `targetRelatedAsset` for each source asset. The query engine orders the related target assets first and uses the first value to order the parent assets. Similar to filtering, the order clause only affects the entities on which it is set. In the example, the `targetRelatedAssets` is not sorted. To sort, you must add another ordering clause on the `Relation` entity.

You should not sort on the target asset node because ordering only makes sense in a collection. If an asset is the source for many relations and the relation has one target asset, you must sort the collection of relations, not the related target asset directly.

The following query example sorts both collections.

Note For simplicity, this query has no filtering. Executing filtering would return all assets and all relations available in Colibra.

JSON

```

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

```

YAML

```

---
ViewConfig:
  Resources:
    Asset:
      Id:
        name: "id"
      Signifier:
        name: "name"
      Relation:
        type: "SOURCE"
      TargetAsset:
        Id:
          name: "targetRelatedAssetId"
        Signifier:
          name: "targetRelatedAsset"
      Order:
        -
          Field:
            name: "targetRelatedAsset"
            order: "ASC"
    Order:
      -
        Field:
          name: "targetRelatedAsset"
          order: "ASC"

```

Differentiate selected properties from properties required in a filter clause

To find the most recently created users, query the `CreatedOn` property and add a filter that uses the `greater than` operator. Adding the `CreatedOn` property to the tree also selects that property.

In cases where you only want the user ID and first and last name, tell the query engine not to return the `CreatedOn` property and use it in the filter.

Note `CreatedOn` is a date expressed as the number of milliseconds since 1/1/1970.

JSON

```
{
  "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" } }
      }
    }
  }
}
```

YAML

```
---
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"
```

Note 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"
      }
    ]
  }
}

```

Strip HTML from text results

Saved values from Collibra also includes HTML formatting tags. Although not visible to users, the user interface uses the tags to format data. These tags are also included when you query data and may look like garbage in Excel reports.

The example below shows how to strip out the HTML formatting tags, leaving only the values.

JSON

```

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

```


YAML

```

---
ViewConfig:
  Resources:
    Community:
      Id:
        name: "communityId"
      Name:
        name: "communityName"
      Description:
        name: "communityDescription"
        stripHtml: true

```

Note Use `stripHtml` on any text field. When true, the returned value is stripped from the HTML tags.

Filtering operators

| Operator | Reverse Operator | Parameters | Type compatibility | Description |
|-------------|------------------|------------|-----------------------|---|
| EQUALS | NOT_EQUALS | 1 | Text, Number, Boolean | Equal/not equal to the value. |
| STARTS_WITH | NOT_STARTS_WITH | 1 | Text | The text starts/does not start with characters. |

| Operator | Reverse Operator | Parameters | Type compatibility | Description |
|-----------------------|-----------------------|------------|-----------------------|---|
| 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. For example, "3, 8" means any digit from 3 to 8 is included. |
| ENDS_ WITH | NOT_ENDS_ WITH | 1 | Text | The text ends/does not end with characters. |
| INCLUDES | NOT_ INCLUDES | 1 | Text | The text contains/does not contain the characters. |
| LESS | GREATER | 1 | Number | The value is strictly less than/greater than the value. |
| LESS_OR_ EQUALS | GREATER_ OR_EQUALS | 1 | Number | The value is less than or equal to/greater than or equal to the value. |
| BETWEEN | / | 2 | Number | The value is included within the 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 values. |

| Operator | Reverse Operator | Parameters | Type compatibility | Description |
|------------------|------------------|--------------|--------------------|---|
| EXISTS | NOT_EXISTS | 1 (optional) | n/a | See below. |
| CR_FILTER_DOMAIN | / | 1 | n/a | ComplexRelation specific filter. Includes only complex relations with at least one related asset in the domain. |

The following table shows samples for each operator.

| Operator | Example |
|-------------------|---|
| EQUALS | <pre>{ "Field": { "name": "domainName", "operator": "EQUALS", "value": "New Business Terms" } }</pre> |
| STARTS_WITH | <pre>{ "Field": { "name": "domainName", "operator": "STARTS_WITH", "value": "New" } }</pre> |
| STARTS_WITH_DIGIT | <pre>{ "Field": { "name": "assetName", "operator": "STARTS_WITH_DIGIT" } }</pre> |
| ENDS_WITH | <pre>{ "Field": { "name": "domainName", "operator": "ENDS_WITH", "value": "Terms" } }</pre> |

| Operator | Example |
|----------------|--|
| INCLUDES | <pre>{ "Field": { "name": "domainName", "operator": "CONTAINS", "value": "Bus" } }</pre> |
| LESS | <pre>{ "Field": { "name": "lastModified", "operator": "GREATER", "value": "1440492290300" } }</pre> |
| LESS_OR_EQUALS | <pre>{ "Field": { "name": "lastModified", "operator": "GREATER_OR_EQUALS", "value": "1440492290300" } }</pre> |
| BETWEEN | <pre>{ "Field": { "name": "lastModified", "operator": "BETWEEN", "values": ["1440492290300", "1440493000000"] } }</pre> |
| NULL | <pre>{ "Field": { "name": "description", "operator": "NULL" } }</pre> |
| IN | <pre>{ "Field": { "name": "statusName", "operator": "IN", "values": ["New", "In Review"] } }</pre> |
| EXISTS | <pre>{ "Field": { "target": "RelationSource", "operator": "EXISTS", "value": "00000000-0000-0000-0000-000000007001", "name": "assetId" } }</pre> |

| Operator | Example |
|------------------|---|
| CR_FILTER_DOMAIN | <pre>{ "Field": { "operator": "CR_FILTER_DOMAIN", "value": "00000000-0000-0000-0000-000000006013" } }</pre> |

EXISTS/NOT_EXISTS filter

In the context of a graph query, the `EXISTS` filter tests the existence of a relationship with another entity. This is the only filter that is explicitly limited to filtering on an entity located directly under the filtered node. To specify which relation should exist/not exist, the filter has a `target` key.

You can also pass a parameter to the `EXISTS` filter. This parameter is used as a secondary filtering element. To query the assets with an attribute of type `Description`, use the `EXISTS` filter on the asset with target value `Attribute` and also the `Id` of the `Description` type in the `value` key of the filter.

The table below lists the possible target values and the expected value type for optional parameters.

| Filtered Entity | Target value | Optional Parameter | Description |
|--------------------------|----------------|--------------------|---|
| Community, Domain, Asset | Responsibility | Role Id | Filter resources related/not related to a responsibility. Optionally, only responsibilities related to the Role Id. |
| Asset | Relation | RelationType Id | Filter assets that are/are not the source or target of a relation. Optionally, only relations related to the RelationType Id. |

| Filtered Entity | Target value | Optional Parameter | Description |
|-----------------|--------------------------|--------------------|---|
| Asset | RelationSource | RelationType Id | Filter assets that are/are not the "source " of a relation. Optionally, only relations related to the RelationType Id. |
| Asset | RelationTarget | RelationType Id | Filter assets that are/are not " target" of a relation. Optionally, only relations related to the RelationType Id. |
| Asset | Attribute | AttributeType Id | Filter assets that have/do not have an attribute. Optionally, only attributes related to the AttributeType Id. |
| Asset | StringAttribute | AttributeType Id | Filter assets that have/do not have a StringAttribute. Optionally, only StringAttributes related to the AttributeType Id. |
| Asset | SingleValueListAttribute | AttributeType Id | Filter assets that have/do not have a SingleValueListAttribute. Optionally, only SingleValueListAttributes related to the AttributeType Id. |

| Filtered Entity | Target value | Optional Parameter | Description |
|--|-------------------------|--------------------|---|
| Asset | MultiValueListAttribute | AttributeTypeId | Filter assets that have/do not have a MultiValueListAttribute. Optionally, only MultiValueListAttribute related to the AttributeTypeId. |
| Asset | BooleanAttribute | AttributeTypeId | Filter assets that have/do not have a BooleanAttribute. Optionally, only BooleanAttributes related to the AttributeTypeId. |
| Asset | NumericAttribute | AttributeTypeId | Filter assets that have/do not have a NumericAttribute. Optionally, NumericAttributes related to the AttributeTypeId. |
| Asset | DateTimeAttribute | AttributeTypeId | Filter assets that have/do not have a DateTimeAttribute. Optionally, only DateTimeAttributes related to the AttributeTypeId. |
| Note The EXISTS/NOT_EXISTS filters are exclusively for communities, domains and assets. | | | |

Filtering in Hierarchy

When the `EQUALS/NOT_EQUALS` and `IN/NOT_IN` operators are 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, relation types or communities. This allows selecting the following assets.

- All assets under a community, including the subcommunities.
- All assets that are of type "X" or one of its subtypes.

Boolean operators

You can combine the filtering operators using Boolean operators. Combining Boolean operators results in a logical binary tree of possibilities. Because the binary tree is not easy to read, the `ViewConfig` provides a way of specifying a `Named Logical Array`.

JSON

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


YAML

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

Note Filtering elements bundled together in a named array, are logically combined using the name of the array: either **AND** or **OR**. You can also nest these logical arrays, allowing all possible Boolean combinations.

JSON

```

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

```

YAML

```

Filter:
  AND:
    -
      OR:
        -
          Field:
            name: "domainId"
            operator: "EQUALS"
            value: "02204077-1cd1-4c70-a7c4-4cd845194b81"
        -
          Field:
            name: "assetId"
            operator: "EXISTS"
            value: "00000000-0000-0000-0000-000000007001"
            target: "RelationSource"
      -
        Field:
          name: "statusName"
          operator: "IN"
          values:
            - "New"
            - "In Review"

```

Filter properties

You can use filter shortcuts to reduce the amount of time required to write a JSON query. For example, `Relation` has a `typeId` parameter that takes an `Id` and eliminates the need to add a `RelationType` node with an `Id` property. These one-line filtering properties are the most commonly used filters because they make the query a lot less verbose.

The following example shows filtering a `StringAttribute` on an `AttributeType` using the `labelId` filtering property.

JSON

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

YAML

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

Refer to [Entities, properties and relations](#) for the list of available filter properties for each entity.

Virtual properties

Collibra does not store virtual properties. It calculates them at runtime and dynamically evaluates the value of each property when the query executes. Virtual properties typically support hierarchical queries that show if the resource has children. Some examples are `hasTaxonomyChildren` and `hasChildForRelation`.

Clarify the relationship between two entities

When two entities are related in more than one way, nesting the entities inside each other is not enough to determine which path to follow. For example, an asset can be either the `source` or `target` of a relation or a user can be the `creator` or the `lastModifier` of a resource. Depending on the entity, there are two possibilities:

- The name of the child entity is changed. For example, `SourceAsset` or `TargetAsset` should be used under `Relation` instead of `Asset`. In this case, they act and behave just like normal assets 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. For example, `Relation` has a `Type` parameter with possible values of `SOURCE` or `TARGET`. This parameter determines the relationship between the `Relation` and the `parentAsset`.

The following example shows the query going two levels deep.

JSON

```
{
  "ViewConfig": {
    "Resources": {
      "Asset": {
        "Id": { "name": "id" },
        "Signifier": { "name": "name" },
        "Relation": {
          "type": "SOURCE",
          "TargetAsset": {
            "Id": { "name": "relatedAssetLevelOneId" },
            "Signifier": { "name": "relatedAssetLevelOne" },
            "Relation": {
              "type": "TARGET",
              "SourceAsset": {
                "Id": { "name": "relatedAssetLevelTwoId" },
                "Signifier": { "name": "relatedAssetLevelTwo" }
              }
            }
          }
        }
      }
    }
  }
}
```

YAML

```

---
ViewConfig:
  Resources:
    Asset:
      Id:
        name: "id"
      Signifier:
        name: "name"
      Relation:
        type: "SOURCE"
      TargetAsset:
        Id:
          name: "relatedAssetLevelOneId"
        Signifier:
          name: "relatedAssetLevelOne"
        Relation:
          type: "TARGET"
        SourceAsset:
          Id:
            name: "relatedAssetLevelTwoId"
          Signifier:
            name: "relatedAssetLevelTwo"

```

These special parameters and custom entity names only exist for a fraction of the available entities. For a complete list, see [Entities, properties and relations](#).

Note To reduce the number of assets returned, the query example above is not filtered. Filtering would return a large amount of data and impact performance.

Page the results

The Output Module also supports paging the results for the root node of the query. You can specify an offset and a length parameter to limit the results to a subset of the complete list.

| 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, you can limit the max count to this value. Passing 0 means no count is done. |

JSON

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

YAML

```

---
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 example query above selects page 3 of all communities, with five results per page.

Note

- Paged results should always be sorted, otherwise the results might seem inconsistent from page to page.
- The paged results list is recalculated upon each request.
- All entities that have been added or removed will appear/disappear from the list, modifying the indexes of the elements in the results list.
- The Collibra Console allows limiting the number of results returned by queries. The values range from 10 000 to 100 000. If enabled, and the limit is set, then:
 - The default `displayLength` value (-1) is overwritten by the limit set through the console.
 - If the `displayLength` set in the `ViewConfig/TableViewConfig` is larger than the limit value set in the Collibra Console, an exception is thrown.

Map the results to a tabular format

The Output Module supports a tabular output format and uses a different kind of `ViewConfig`, called `TableViewConfig`. `TableViewConfig` has a `Columns` mapping

section that assigns each selected field to a column. The previous examples use the `ViewConfig` as input to the API to produce a JSON tree format.

The following example uses `TableViewConfig`. This is available under the same `{{domain}}/rest/2.0/outputModule/export/json` endpoint, just using the `TableViewConfig` as the JSON payload.

JSON

```
{
  "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" } }
    ]
  }
}
```

YAML

```

---
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"

```

When formatted, this query 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": ""
    }
]
}

```

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

In the following example, the `"displayLength"` value is set to 0. This query shows the number of entities without retrieving actual results.

Note The JSON Data Table output contains the total number of available records in Collibra for this query, which is `iTotalDisplayRecords`. It also contains the number of records returned in this set, which is `iTotalRecords`.

```

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

```

You can use the `TableViewConfig` queries with the following endpoints:

- `{{domain}}/rest/2.0/outputModule/export/{{json | csv}}`
- `{{domain}}/rest/2.0/outputModule/export/{{json | csv | excel}}-file`

- `{{domain}}/rest/2.0/outputModule/export/{{json | csv | excel}}-job`

Handling to-many results in a tabular format

You can select all assets from a domain together with their `Note` attributes. Each asset may have multiple notes. When there are multiple notes, the most recent note should be ordered at the top of the list.

The `TableViewConfig` may look similar to the example below.

JSON

```
{
  "TableViewConfig": {
    "Resources": {
      "Asset": {
        "Id": { "name": "assetId" },
        "Signifier": { "name": "assetName" },
        "StringAttribute": {
          "LongExpression": { "name": "note" },
          "CreatedOn": { "name": "noteCreatedOn" },
          "Order": [ { "Field": { "name": "noteCreatedOn" },
"order": "DESC" } } ]
        },
        "Domain": {
          "Id": { "name": "domainId" }
        },
        "Filter": { "Field": { "name": "domainId", "operator":
"EQUALS", "value": "f342423f-54fd-4643-935b-adbd9e7f5e25" } },
        "Order": [ { "Field": { "name": "assetName" } } ]
      }
    },
    "Columns": [
      { "Column": { "fieldName": "assetId" } },
      { "Column": { "fieldName": "assetName" } },
      { "Column": { "fieldName": "note" } }
    ]
  }
}
```

YAML

```

---
TableViewConfig:
  Resources:
    Asset:
      Id:
        name: "assetId"
      Signifier:
        name: "assetName"
      StringAttribute:
        LongExpression:
          name: "note"
        CreatedOn:
          name: "noteCreatedOn"
      Order:
        -
          Field:
            name: "noteCreatedOn"
            order: "DESC"
      Domain:
        Id:
          name: "domainId"
      Filter:
        Field:
          name: "domainId"
          operator: "EQUALS"
          value: "f342423f-54fd-4643-935b-adbd9e7f5e25"
      Order:
        -
          Field:
            name: "assetName"
    Columns:
      -
        Column:
          fieldName: "assetId"
      -
        Column:
          fieldName: "assetName"
      -
        Column:
          fieldName: "note"

```

Depending on the format requested, the results might be different. In Excel or CSV format, each asset is duplicated on a new row for each note value.

| | A | B | C | D |
|---|--------------------------------------|-----------------|--------------------|---|
| 1 | termId | 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 | | | | |

This is similar to using SQL queries to join two tables with a one-to-many relationship. Unlike SQL, if you select an asset with two notes and three responsibilities, the asset 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, does not add duplicate rows to the results. Instead, it returns the first note found and discards the other notes.

Example "First note" is missing for "Business Asset 1"

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

For tabular formats that do not duplicate rows, you can add the `Group` mapping construct to the `Columns` section.

JSON

```

{
  "TableViewConfig": {
    "Resources": {
      "Asset": {
        "Id": { "name": "assetId" },
        "Signifier": { "name": "assetName" },
        "StringAttribute": {
          "LongExpression": { "name": "note" },
          "CreatedOn": { "name": "noteCreatedOn" },
          "Order": [ { "Field": { "name": "noteCreatedOn",
"order": "DESC" } } ]
        },
        "Domain": {
          "Id": { "name": "domainId" }
        },
        "Filter": { "Field": { "name": "domainId", "operator":
"EQUALS", "value": "f342423f-54fd-4643-935b-adbd9e7f5e25" } },
        "Order": [ { "Field": { "name": "assetName" } } ]
      }
    },
    "Columns": [
      { "Column": { "fieldName": "assetId" } },
      { "Column": { "fieldName": "assetName" } },
      {
        "Group": {
          "name": "Notes",
          "Columns": [
            { "Column": { "fieldName": "note" } }
          ]
        }
      }
    ]
  }
}

```

YAML

```

---
TableViewConfig:
  Resources:
    Asset:
      Id:
        name: "assetId"
      Signifier:
        name: "assetName"
      StringAttribute:
        LongExpression:
          name: "note"
      CreatedOn:
        name: "noteCreatedOn"
      Order:
        -
          Field:
            name: "noteCreatedOn"
            order: "DESC"
      Domian:
        Id:
          name: "domainId"
      Filter:
        Field:
          name: "domainId"
          operator: "EQUALS"
          value: "f342423f-54fd-4643-935b-adbd9e7f5e25"
      Order:
        -
          Field:
            name: "assetName"
    Columns:
      -
        Column:
          fieldName: "assetId"
      -
        Column:
          fieldName: "assetName"
      -
        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": [
    {
      "assetId": "c20d5b39-6c5d-411b-adcb-82a1dd3851cc",
      "assetName": "Business Term 1",
      "Notes": [
        {
          "note": "Second Note"
        },
        {
          "note": "First note"
        }
      ]
    },
    {
      "assetId": "1a6a8f73-43b0-4a29-84c3-baaa3467be70",
      "assetName": "Business Term 2",
      "Notes": [
        {
          "note": "Single note on BT2"
        }
      ]
    },
    {
      "assetId": "7329349e-0631-41a7-a740-738979d887c6",
      "assetName": "Business Term 3",
      "Notes": [
        {
          "note": "Single Note on BT3"
        }
      ]
    }
  ]
}
```

Note

Here are 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.

Set an execution timeout

Queries that run on complicated or large amounts of data may be slower than expected. Usually, the best approach is to paginate the results. In cases where the complexity or amount of data is unknown, a timeout can break up the execution. The Output Module can timeout, not only on the execution logic level, but also break running database queries to protect the database load from stress.

You can set a timeout for each `ViewConfig` and `TableViewConfig` execution on the main config level. Defining it in the body of the query is optional.

If a timeout is not set in the `ViewConfig` or `TableViewConfig`, then a default value is added. You can configure the default value in the Colibra console, the default setting is eight hours.

Warning

- No single query may run longer than 24 hours, which is the maximum value.
- Pagination is recommended for queries that may run longer.
- Those values will significantly smaller in the next major release, so it would be prudent to think about pagination.
- If the `queryTimeout` is more than 24 hours, the system will overwrite it with the maximum 24-hour limit value.
- Important exceptions are the `{{domain}}/rest/2.0/outputModule/export/{{csv | excel}}-job` endpoints. Here, data is calculated in chunks, with the size of the chunk defined in the Colibra Console. A separate query calculates each chunk and the timeout value set in the `TableViewConfig` will be a timeout value calculation for that 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 if the execution takes longer than the given number of seconds. |

Example of ViewConfig with a timeout set:

JSON

```
{
  "ViewConfig": {
    "queryTimeout": 5,
    "Resources": {
      "Domain": {
        "name": "d",
        "Name": {
          "name": "vocName"
        },
        "Asset": {
          "name": "t",
          "Signifier": {
            "name": "assetName"
          },
          "AssetType": {
            "name": "tt",
            "Name": {
              "name": "assetType"
            }
          }
        }
      }
    }
  }
}
```

YAML

```

---
ViewConfig:
  queryTimeout: 5
  Resources:
    Domain:
      name: "d"
      Name:
        name: "vocName"
    Asset:
      name: "t"
      Signifier:
        name: "assetName"
      AssetType:
        name: "tt"
      Name:
        name: "assetType"

```

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

Structural validation of the query

Because writing `ViewConfigs` and `TableViewConfigs` is a tedious and error-prone task, the following endpoints allow using the `validationEnabled` parameter.

- `{{domain}}/rest/2.0/outputModule/export/{{xml | json | csv}}`
- `{{domain}}/rest/2.0/outputModule/export/{{xml | json | csv | excel}}-file`
- `{{domain}}/rest/2.0/outputModule/export/{{xml | json | csv | excel}}-job`

This parameter, when set to true, enables validation of the input

`ViewConfig/TableViewConfig`. By default, the parameter value is set to false.

The example below shows a small typo in the filter. `userID` is used instead of `userId`. When you make a POST request to

{{domain}}/rest/2.0/outputModule/export/json?validationEnabled=true,
the following body results.

JSON

```
{
  "ViewConfig": {
    "displayLength": 5,
    "Resources": {
      "Community": {
        "Id": { "name": "communityId" },
        "Name": { "name": "community" },
        "Responsibility": {
          "Id": { "name": "responsibilityId" },
          "User": {
            "Id": { "name": "userId" },
            "FirstName": { "name": "userName" }
          }
        }
      },
      "Filter": { "Field": { "name": "userId", "Operator": "NOT_"
NULL" } }
    }
  }
}
```

YAML

```

---
ViewConfig:
  displayLength: 5
  Resources:
    Community:
      Id:
        name: "communityId"
      Name:
        name: "community"
      Responsibility:
        Id:
          name: "responsibilityId"
        User:
          Id:
            name: "userId"
          FirstName:
            name: "userName"
    Filter:
      Field:
        name: "userId"
        Operator: "NOT_NULL"

```

The response will be similar to the example below.

```

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

```

API endpoints and query formats

The available rest API endpoints URL are:

- `{{domain}}/rest/2.0/outputModule/export/{{format}}`
- `{{domain}}/rest/2.0/outputModule/export/{{format}}-file`
- `{{domain}}/rest/2.0/outputModule/export/{{format}}-job`

The available formats are XML, JSON, CSV and Excel.

In this chapter

| | |
|--|----|
| Endpoints and formats | 67 |
| ViewConfig/TableViewConfig and formats | 67 |
| Single query and multi-query | 67 |

Endpoints and formats

| Endpoint | CSV | JSON | CSV | EXCEL |
|---|-----|------|-----|-------|
| • {{domain}}/rest/2.0/outputModule/export/{{format}} | YES | YES | YES | NO |
| • {{domain}}/rest/2.0/outputModule/export/{{format}}-file | YES | YES | YES | YES |
| • {{domain}}/rest/2.0/outputModule/export/{{format}}-job | YES | YES | YES | YES |

ViewConfig/TableViewConfig and formats

| Format | Supports ViewConfig | Supports TableViewConfig |
|--------|---------------------|--------------------------|
| XML | YES | NO |
| JSON | YES | YES |
| CSV | NO | YES |
| EXCEL | NO | YES |

Single query and multi-query

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

| Endpoint | CSV | JSON | CSV | EXCEL |
|--|--------|--------|--------|--------|
| {{domain}}/rest/2.0/outputModule/export/{{format}} | SINGLE | SINGLE | SINGLE | SINGLE |

| Endpoint | CSV | JSON | CSV | EXCEL |
|---|--------|--------|--------|--------|
| {{domain}}/rest/2.0/outputModule/export/{{format}}-file | SINGLE | SINGLE | SINGLE | SINGLE |
| {{domain}}/rest/2.0/outputModule/export/{{format}}-job | SINGLE | SINGLE | MULTI | MULTI |

Entities, properties and relations

| Entity | | |
|---|-----------|---------------------------------------|
| Entity is the base abstract class of all other entities. An abstract entity cannot be queried, thus <code>Entity</code> 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 the following properties and relations. An abstract entity cannot be queried, thus <code>Resource</code> cannot be used in the query tree. | | |
| Properties | | |
| createdOn | Number | Creation date (# milliseconds since 1/1/1970). |
| createdOnTimestamp | 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). |
| lastModifiedTimestamp | 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 | <ul style="list-style-type: none"> 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 Asset. All assets share 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 the representation. |
| Domain | Many-to-One | The domain containing the representation. |
| AssetType | Many-to-One | The AssetType of the representation. |
| Attribute | One-to-Many | The collection of attributes in the representation. |
| StringAttribute | One-to-Many | The collection of StringAttributes in the representation. |
| ScriptAttribute | One-to-Many | The collection of ScriptAttributes in the representation. |
| SingleValueListAttribute | One-to-Many | The collection of SingleValueListAttributes in the representation. |
| MultiValueListAttribute | One-to-Many | The collection of MultiValueListAttributes in the representation. |
| BooleanAttribute | One-to-Many | The collection of BooleanAttributes in the representation. |

| | | |
|-------------------|-------------|---|
| NumericAttribute | One-to-Many | The collection of NumericAttributes in the representation. |
| DateTimeAttribute | One-to-Many | The collection of DateTimeAttributes in the representation. |
| DateAttribute | One-to-Many | The collection of DateAttributes in the representation. |

Organization

Extends Resource

Represents the hierarchy of organizations available in Collibra.

Properties

| | | |
|--------------------|------------|---|
| name | Text (255) | The name of the organization. |
| description | Text | The description of the organization. |
| uri | Text (255) | The URI of the organization. |
| language | Text(255) | The name of the language used. |
| meta | Boolean | Indicates if the community is related to the meta model, such as a hidden organization. |
| hasNonMetaChildren | Boolean | Indicates if the organization contains non-meta subcommunities or domains. |

| | | |
|---------------------------|-------------|---|
| hasNonMetaChildCommunity | Boolean | Indicates if the organization contains non-meta communities. |
| organizationType | Text | Indicates if the organization is a community ("C") or a domain ("D") |
| Relations | | |
| ParentCommunity | Many-to-One | The parent community of this organization. Null for root communities. Optional. |
| Community | One-to-Many | The collection of subcommunities. |
| Domain | One-to-Many | The collection of vocabularies contained in the organization. |
| Responsibility | One-to-Many | The collection of responsibilities playing a role in the organization. |
| SubCommunities | One-to-Many | The collection of domains contained in the community. |
| Comment | One-to-Many | The collection of comments contained in the community. |
| Asset | One-to-Many | The collection of assets contained in the community. |
| DomainType | One-to-Many | The type of domain. |
| Mapping | One-to-Many | The collection of mappings corresponding to this domain. |
| Filtering Property | | |

| | | |
|---------------|---------|--|
| rootCommunity | Boolean | When true, the query engine adds a filter retaining only root communities. Only available when the community is also root of the query tree. |
|---------------|---------|--|

Community

Extends Organization

Exact synonym of an organization but with default filtering on organizationType equal to "C"

ParentCommunity

Extends Community

Exact synonym of a community. It can only be used as a child of the community to disambiguate the relationship followed.

Domain

Extends Organization

Synonym of an organization but with default filtering on organizationType equal to "D" and with overridden relation for Community

Relations

| | | |
|-----------|-------------|-----------------------|
| Community | Many-to-One | The parent community. |
|-----------|-------------|-----------------------|

DomainType

Extends Resource

Each domain has a `DomainType`.

Properties

| | | |
|-------------|------------|--|
| signifier | Text (255) | The name of the <code>DomainType</code> . |
| name | | Synonym for signifier. |
| description | Text | The description of the <code>DomainType</code> . |
| meta | Boolean | Indicates if the <code>DomainType</code> is related to the Col-libra meta model. |

Relations

| | | |
|------------------|-------------|--|
| Domain | One-to-Many | The collection of domain instances of the <code>DomainType</code> . |
| DomainType | Many-to-One | The parent <code>DomainType</code> of the <code>DomainType</code> . Null for root <code>DomainTypes</code> . Optional. |
| ChildDomainTypes | One-to-Many | The collection of <code>DomainType</code> children. |

ChildDomainTypes

Extends `DomainType`
Collection of `DomainType`

Exact synonym of `DomainType`. Can only be used as a child of `DomainType` to disambiguate the relationship followed.

| <h1>RelationType</h1> <p>Extends Resource</p> | | |
|--|-------------|---|
| <p>A <code>RelationType</code> defines a class of relationship between two <code>AssetTypes</code>, also called <code>AssetTypes</code>.</p> | | |
| 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 <code>RelationType</code> . |
| Relations | | |
| Relation | One-to-Many | The collection of relation instances with this <code>RelationType</code> . |
| SourceAssetType | Many-to-One | The <code>AssetType</code> that is head of the <code>RelationType</code> . <code>SourceAssetType</code> is a synonym of <code>AssetType</code> and clarifies which path is followed from the <code>Relation</code> entity to its child. In this case, the child node is the head. |
| TargetAssetType | Many-to-One | The <code>AssetType</code> that is the tail of the <code>RelationType</code> . <code>TargetAssetType</code> is a synonym of <code>AssetType</code> and clarifies which path is followed from the <code>Relation</code> 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 <code>AssetType</code> entity to the <code>RelationType</code> . The possible values are either <code>HEAD</code> or <code>TAIL</code> , which tells whether the parent <code>AssetType</code> is the head or the tail of the <code>RelationType</code> . The default value is <code>HEAD</code> . |

| Relation | | |
|---|-------------|--|
| Extends Resource | | |
| A <code>Relation</code> links two <code>Assets</code> 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 | | |
| RelationType | Many-to-One | The type of this relation. |
| SourceAsset | Many-to-One | The source asset of this relation. |
| TargetAsset | Many-to-One | The target asset of this relation. |
| Parent relationship selector. Only if the parent is a asset node or is of type inheriting from an asset node. | | |

| | |
|---------------------------|--|
| type | This parameter allows specifying which path should be followed from the parent asset entity to this relation. The possible values are either <code>SOURCE</code> or <code>TARGET</code> , which tells whether the parent asset is the source or target of the relation. This parameter is mandatory because there is no default value. |
| Filtering Property | |
| typeld | Allows filtering relations using the Id value of their related <code>RelationType</code> . |

ComplexRelation

Extends Asset

A `ComplexRelation` is an anonymous asset, whose signifier, or name, has been generated.

Properties

/

Relations

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

Filtering Property

| | |
|--------|--|
| typeld | Allows filtering ComplexRelations using the Id value of their related <code>ComplexRelationType</code> . |
|--------|--|

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 AssetType | | |
| A <code>ComplexRelationType</code> determines the type of a <code>ComplexRelation</code> . | | |
| Properties | | |
| / | | |
| Relations | | |
| ComplexRelation | OneToMany | The collection of <code>ComplexRelation</code> instances with the <code>ComplexRelationType</code> . |
| ComplexRelationLegType | OneToMany | The collection of <code>ComplexRelationLegTypes</code> linked to the <code>ComplexRelationType</code> . |
| ComplexRelationAttributeType | OneToMany | The collection of <code>ComplexRelationAttributeTypes</code> linked to the <code>ComplexRelationType</code> . |

ComplexRelationLegType

Extends Resource

A `ComplexRelationLegType` is a `RelationType` used in the context of a `ComplexRelationType`. The `SourceAssetType` of those `RelationTypes` of the `ComplexRelationType`. It can only be used as a child of `ComplexRelationType`.

Properties

| | | |
|----------|--------|---|
| min | Number | The minimum occurrences of this <code>RelationType</code> in the <code>ComplexRelationType</code> . |
| max | Number | The maximum occurrences of this <code>RelationType</code> in the <code>ComplexRelationType</code> . |
| legOrder | Number | Order of this <code>ComplexRelationLegType</code> in the <code>ComplexRelationType</code> . |

Relations

| | | |
|---------------------------|-------------|--|
| <code>RelationType</code> | Many-to-One | The <code>RelationType</code> of the <code>ComplexRelationLegType</code> . |
|---------------------------|-------------|--|

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 ComplexRelationType. |
| max | Number | The maximum occurrences of this AttributeType in the ComplexRelationType. |
| readOnly | Boolean | Indicates 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 ComplexRelationAttributeType. |

| Asset | | |
|---|-------------|---|
| Extends Representation | | |
| An <code>Asset</code> is the basic building block capturing information about the assets available in Collibra. | | |
| Properties | | |
| signifier | Text (2000) | The full name of the asset. |
| displayName | Text (2000) | The display name of the asset. |
| articulationScore | Number | Result of the last calculation of the articulation score. |

| | | |
|--|-------------|--|
| hasChildrenForRelation (deprecated) | Boolean | <p>Virtual calculated property indicating if this asset has children for the relation type defined at the query level. This property takes two additional parameters:</p> <ul style="list-style-type: none"> • <code>the RelationType</code> • <code>direction</code> (role or co-role) <p>For example:</p> <pre>"HasChildrenForRelation": { "name": "hasChildren", "relationTypeId": "00000000-0000-0000-0000- 000000007005", "roleDirection": true }</pre> <p>It can only be used if <code>Asset</code> is a root node of the query. It is not inherited by nodes extending the <code>Asset</code> node.</p> |
| avgRating | Number | Average value of all ratings assigned to the asset. |
| ratingsCount | Number | Number of all ratings signed to the asset. |
| class | Text | With other entities that extend the asset, can be used to differentiate amongst the various subclasses. |
| Relations | | |
| Relation | One-to-Many | The collection of relations this asset has. See <code>Relation</code> for a mandatory type parameter. |

| | | |
|---------------------------|---|--|
| Responsibility | One-to-Many | The collection of responsibilities this asset has. |
| Mapping | One-to-Many | The related mappings. |
| Tag | Many-to-Many | The collection of tags associated with this asset. |
| Filtering Property | | |
| rootOfRelation | <p>An array relation types/direction pairs. Root assets are not the child of any of the relations.</p> <p>For example:</p> <pre> "rootOfRelation": [{ "relationTypeId": "000000000-0000-0000-0000-000000007038", "roleDirection": true }, { "relationTypeId": "000000000-0000-0000-0000-000000007005", "roleDirection": true }], </pre> | |

SourceAsset

Extends Asset

Exact synonym of `Asset`. It can only be used as a child of relation to disambiguate the relationship followed.

TargetAsset

Extends Asset

Exact synonym of `Asset`. It can only be used as a child of a relation to disambiguate the relationship followed.

SourceAssetType

Extends `AssetType`

Exact synonym of `AssetType`. It can only be used as a child of `RelationType` to disambiguate the relationship followed.

TargetAssetType

Extends `AssetType`

Exact synonym of `AssetType`.

Can only be used as a child of `RelationType` to disambiguate the relationship followed.

AssetType

Extends Resource

A `AssetType`, also called `AssetType`, determines the type of asset, which is an `Asset`

Properties

signifier

Text (255)

The name of this `AssetType`.

| | | |
|--------------------|-------------|--|
| name | | Synonym for signifier. |
| description | Text | The description of the <code>AssetType</code> . |
| meta | Boolean | Is the <code>AssetType</code> related to the Collibra meta model. |
| color | Text | The color of the <code>AssetType</code> . |
| icon | Text | The icon of the <code>AssetType</code> . |
| acronym | Text | The acronym of the <code>AssetType</code> |
| symbolType | Text | Defines the icon or acronym used in Collibra. Possible values are: <code>ICON</code> , <code>ACRONYM</code> and <code>NONE</code> . |
| displayNameEnabled | Boolean | Indicates if the display name is enabled for all assets of this <code>AssetType</code> . |
| ratingEnabled | Boolean | Are ratings enabled for all assets of this <code>AssetType</code> . |
| Relations | | |
| Asset | One-to-Many | The collection of instances of this <code>AssetType</code> . |
| AssetType | Many-to-One | The parent <code>AssetType</code> of this <code>AssetType</code> . |
| ChildAssetTypes | One-to-Many | The collection of concept types that have this <code>AssetType</code> as parent. |

ChildAssetTypes

Extends `AssetType`
Collection of `AssetType`

Can only be used as a child of `AssetType` to disambiguate the relationship followed.

The `ComplexRelationType`, despite inheriting from `AssetType`, does not support `ChildAssetTypes` node.

Attribute

Extends `Resource`

Attribute represents an attribute linked to a representation.

Properties

| | | |
|-------|------|---|
| value | Text | The text value of this attribute. |
| class | Text | With other entities, extends attribute. You may use the <code>class</code> qualifier to differentiate between the various subclasses. |

Relations

| | | |
|---------------|-------------|---|
| AttributeType | Many-to-One | The type of attribute. |
| Asset | Many-to-One | The asset to which the attribute belongs. |

Filtering Property

| | |
|---------|--|
| labelId | Allows filtering the attributes based on the <code>Id</code> of their related <code>AttributeType</code> . |
|---------|--|

StringAttribute

Extends Attribute

A `StringAttribute` is an attribute dedicated to text values.

Properties

| | | |
|-----------------------------|------|--|
| <code>longExpression</code> | Text | The unbounded text value. Obsolete, but returns the same content as <code>Attribute:value</code> . |
|-----------------------------|------|--|

ScriptAttribute

Extends Attribute

A `ScriptAttribute` is an attribute dedicated to script values.

Properties

| | | |
|---------------------|------|--|
| <code>script</code> | Text | The script. Obsolete, but returns the same content as <code>Attribute:value</code> . |
|---------------------|------|--|

SingleValueListAttribute

Extends Attribute

A `SingleValueListAttribute` is an attribute dedicated to storing a single value selected from a list.

MultiValueListAttribute

Extends Attribute

A `MultiValueListAttribute` is an attribute dedicated to storing 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

| | | |
|-----------------------|--------|--|
| <code>dateTime</code> | 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

| | | |
|------------------------|--------|--|
| <code>date</code> | Number | The date value expressed as the number of milliseconds since 1/1/1970. |
| <code>timestamp</code> | 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 the <code>AttributeType</code> . |
| name | | Synonym for signifier. |
| description | Text | The description of this <code>AttributeType</code> . |
| attributeKind | Text(255) | The <code>AttributeType</code> kind. The possible values are: BOOLEAN, STRING, NUMERIC, DATE, DATE_TIME, SINGLE_VALUE_LIST, MULTI_VALUE_LIST and SCRIPT. |
| language | Text(255) | The name of the language used. The kind is SCRIPT. |
| isInteger | Boolean | Indicates if the <code>AttributeType</code> defines an integer or decimal. If true, it defines an integer. If false, it defines a decimal. The kind is NUMERIC. |
| allowedValues | Text | Comma separated list of values. The kind is SINGLE_VALUE_LIST or MULTI_VALUE_LIST. |
| Relations | | |
| Attribute | One-to-Many | The collection of <code>Attributes</code> of this type |

| | | |
|--|------|----------------|
| <h1>User</h1> <p>Extends Resource</p> | | |
| <p>Represents Collibra users. Any resource has a creation date and the last modification date. Collibra also stores which user made each of these operations. The User entity is related to all types as the creator and/or last modifier of the entity.</p> | | |
| Properties | | |
| userName | Text | The user name. |

| | | |
|-------------------------|--------------|--|
| firstName | Text | The first name. |
| lastName | Text | The last name. |
| fullName | Text | Virtual property containing the first and last name together, which is useful for filters. |
| gender | Text | The gender. |
| language | Text | The user language. |
| activated | Boolean | Indicates if the user is activated. |
| ldapUser | Boolean | Indicates if the user is a LDAP User. |
| apiUser (deprecated) | Boolean | Indicates if this is an API user. |
| enabled | Boolean | Indicates if the user is enabled. |
| emailAddress | Text | The user's primary email address. |
| guest | Boolean | Indicates if this is a guest user. |
| Relations | | |
| Email | Many-to-Many | The collection of emails owned by the user. |
| Phone | Many-to-Many | The collection of phone numbers owned by the user. |
| InstantMessagingAccount | Many-to-Many | The collection of <code>InstantMessagingAccount</code> accounts owned by this user. |

| | | |
|-----------------|--------------|---|
| Website | Many-to-Many | The collection of websites owned by the user. |
| Address | Many-to-Many | The collection of addresses owned by the user. |
| Community | One-to-Many | The collection of communities created or last modified by the user. |
| Domain | One-to-Many | The collection of vocabularies created or last modified by the user. |
| DomainType | One-to-Many | The collection of <code>DomainTypes</code> created or last modified by the user. |
| RelationType | One-to-Many | The collection of <code>RelationType</code> created or last modified by the user. |
| Relation | One-to-Many | The collection of relations created or last modified by the user. |
| ComplexRelation | One-to-Many | The collection of <code>ComplexRelations</code> created or last modified by the user. |
| Asset | One-to-Many | The collection of assets created or last modified by the user. |
| AssetType | One-to-Many | The collection of <code>AssetTypes</code> created or last modified by the user. |
| Attribute | One-to-Many | The collection of attributes created or last modified by the user. |

| | | |
|--------------------------|-------------|--|
| StringAttribute | One-to-Many | The collection of <code>StringAttributes</code> created or last modified by the user. |
| ScriptAttribute | One-to-Many | The collection of <code>ScriptAttributes</code> created or last modified by the user. |
| SingleValueListAttribute | One-to-Many | The collection of <code>SingleValueListAttributes</code> created or last modified by the user. |
| MultiValueListAttribute | One-to-Many | The collection of <code>MultiValueListAttributes</code> created or last modified by the user. |
| BooleanAttribute | One-to-Many | The collection of <code>BooleanAttributes</code> created or last modified by the user. |
| NumericAttribute | One-to-Many | The collection of <code>NumericAttributes</code> created or last modified by the user. |
| DateTimeAttribute | One-to-Many | The collection of <code>DateTimeAttributes</code> created or last modified by the user. |
| DateAttribute | One-to-Many | The collection of <code>DateAttributes</code> created or last modified by the user. |
| AttributeType | One-to-Many | The collection of <code>AttributeTypes</code> 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 to which this user belongs. |
| Responsibility | One-to-Many | The collection of responsibilities linking this user to a role on an asset, domain 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 the user. |
| WorkflowTaskInfo (deprecated) | One-to-Many | The collection of <code>WorkflowTaskInfos</code> created or last modified by the user. |
| Mapping | One-to-Many | The collection of mappings created or last modified by the user. |
| Parent Relationship Selector | | |

| | |
|----------|---|
| linkType | This parameter allows specifying the path that should be followed from the parent resource to a user. When the parent resource is responsibility or group, linkType is not used and the relationship defined for responsibility or group is used. When a user is the parent node, linkType determines the relationship with the child resources that have a created or last modified kind of relationship. See relations above. 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 turns into a simple CREATED when User is not the root of the query. |
|----------|---|

Email

Extends Resource

Email represents one of the user's email addresses. It can only be used as a child of the user ser.

Properties

| | | |
|--------------|------|--------------------|
| emailAddress | Text | The email address. |
|--------------|------|--------------------|

Phone

Extends Resource

Phone represents one of the user's phone numbers. It can only be used as a child of the user.

Properties

| | | |
|-------------|------|--|
| phoneNumber | Text | The phone number. |
| phoneType | Text | The phone type: FAX, MOBILE, OTHER, PAGER, PRIVATE and WORK. |

InstantMessagingAccount

Extends Resource

`InstantMessagingAccount` represents one of the user's instant messaging account. It can only be used as a child of the user.

Properties

| | | |
|-----------------------------|------|--|
| account | Text | The account id |
| instantMessagingAccountType | Text | The instant messaging type: AOL, GTALK, ICQ, JABBER, LIVE_MESSENGER, SKYPE or YAHOO_MESSENGER. |

Website

Extends Resource

Website represents one of the user's websites. It can only be used as a child of the user.

Properties

| | | |
|-------------|------|---|
| url | Text | The URL of the website. |
| websiteType | Text | The type of website: FACEBOOK, LINKEDIN, MYSPACE, TWITTER or WEBSITE. |

Address

Extends Resource

Address represents one of the user's addresses. It can only be used as a child of the 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. |
| addressType | Text | The address type: HOME or 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 that are part of this group. |
| Responsibility | One-to-Many | The collection of responsibilities linking this group to a role on an asset, domain or community. |

| Responsibility | | |
|---|-------------|---|
| Extends Resource | | |
| A responsibility links a user or group with a role on an asset, domain 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. |
| Asset | Many-to-One | The associated asset. |
| Domain | Many-to-One | The associated domain. |
| Community | Many-to-One | The associated community. |
| Filtering Property | | |

| | |
|--------|--|
| roleId | Allows filtering responsibilities using the Id property of the related role. |
|--------|--|

| |
|--|
| <h2>Role</h2> <p>Extends Asset (deprecated)</p> |
| The Role that a user plays. For example, Steward or Admin. |

| | | |
|---|-------------|----------------------------|
| <h1>Status</h1> <p>Extends Resource</p> | | |
| The status of an asset. | | |
| Properties | | |
| signifier | Text(255) | The name of the status. |
| description | Text | The status description. |
| Relations | | |
| Asset | One-to-Many | The assets of this status. |

| |
|--|
| <h2>WorkflowTaskInfo (deprecated)</h2> <p>Extends Resource</p> |
| WorkflowTaskInfo holds all information 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. |
| itemVerbalized | Text | The verbalized version of the related item. |
| taskType | Text | The type of task. |
| assignee | Text | The id of the assigned user. |
| candidateUsers | Text | The ids of candidate users. |
| domain | Text | The related domain Id. |
| community | Text | The related community Id. |
| status | Text | The status of the task. |

Mapping

Extends Resource

A **Mapping** links an externally defined entity, such as an asset or domain, to one entity.

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 synchronization date. |
| syncAction | Text | The last synchronization action: ADD, UPDATE or REMOVE. |
| description | Text | Description of this mapping. |
| Relations | | |
| Asset | Many-to-One | The related asset. |
| Domain | Many-to-One | The related domain. |

| | | |
|---|--------------|----------------------|
| <h2>Tag</h2> <p>Extends Resource</p> | | |
| A <code>Tag</code> allows categorizing assets by adding one or more labels. | | |
| Properties | | |
| name | Text | The name of the tag. |
| Relations | | |
| Asset | Many-to-Many | The related assets. |

| | | |
|--|--|--|
| <h2>DataQualityRule (deprecated)</h2> <p>Extends Resource</p> | | |
| A <code>DataQualityRule</code> 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 <code>Scope</code> describes the scope of an assignment. | | |
| Properties | | |
| name | Text | The name of the scope. |
| description | Text | The description of the scope. |
| Relations | | |

| Comment | | |
|--|--------|---|
| Extends Resource | | |
| Comment represents a single comment of a resource. | | |
| Properties | | |
| content | String | The content of this comment. |
| resourceType | String | A type of the resource to which this comment belongs. |

| Relations | | |
|--------------------|-------------|---|
| ParentComment | Many-to-One | The parent comment of this comment. |
| Comment | One-to-Many | List of subcomments of this comment |
| Asset | One-to-One | The asset to which this comment is linked. |
| Domain | One-to-One | The domain to which this comment is linked. |
| Community | One-to-One | The community to which this comment is linked. |
| Filtering property | | |
| rootComment | Boolean | When true, the query engine adds a filter retaining only root comments. |

ParentComment

Extends Resource

`ParentComment` can only be used as a child of a comment to disambiguate the relationship followed.

DataType (deprecated)

Extends Entity

A `DataType` is a Catalog entity that characterizes a data element's data type.

Properties

| | | |
|-----------------|------|---|
| name | Text | The name of the type: Date or SSN. |
| description | Text | Description of the type. |
| class | Text | The class of <code>DataTypes</code> : BASE and ADVANCED. |
| logicalDataType | Text | The corresponding logical data type used by the profiling job. It is one of the base types. |

Relations

| | | |
|---------------|-------------|--|
| DataTypeMatch | One-to-Many | The related <code>DataTypeMatches</code> holding a specific percentage of match value for a Data Element instance. |
|---------------|-------------|--|

AdvancedDataType (deprecated)

Extends `DataType`

An `AdvancedDataType` is an extension of one of the base `DataTypes`, for example, Text, Numeric or Date, that provides patterns that help 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 <code>AdvancedDataType</code> . |
|------------------|-------------|---|

DataTypeMatch (deprecated)

Extends Entity

A `DataTypeMatch` contains profiling results indicating the percentage of the actual data behind a `DataElement` asset that matches a `DataType`.

Properties

| | | |
|------------|--------|--------------------------|
| percentage | Double | The matching percentage. |
|------------|--------|--------------------------|

Relations

| | | |
|----------|-------------|-------------------------------------|
| Asset | Many-to-One | The related Data Element. |
| DataType | Many-to-One | The matched <code>DataType</code> . |

BaseView (deprecated)

Extends Resource

An abstract entity base class of `View` and `DiagramPicture`.

Properties

| | | |
|--------------|---------|--|
| name | Text | The name of the <code>baseView</code> . |
| description | Text | The description of the <code>baseView</code> . |
| config | Text | The JSON config of the <code>baseView</code> . |
| originalView | Text | The Id of the <code>originalView</code> of this base view, meaning the view from which this base view was created. |
| isDefault | Boolean | Indicates if this is a default <code>baseView</code> . |
| isPreferred | Boolean | Indicates if this a preferred pinned <code>baseView</code> |

Relations

View (deprecated)

Extends BaseView

A view in Collibra.

Properties

Relations

DiagramPicture (deprecated)

Extends BaseView

A diagram illustration.

Properties

| | | |
|-----|------|---|
| svg | Text | Text field containing an SVG representation of the diagram picture. |
|-----|------|---|

Relations

| | | |
|---------------------------|--------------|--|
| View | Many-to-One | The view used to create or 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 `DiagramPicture` sharing rule. A diagram picture can be shared with a user, group or 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, only exposed to the graph query engine to show the asset linked to a `DiagramPicture`.

Properties

Relations

| | | |
|-------|-------------|--------------------------------|
| Asset | Many-to-One | The asset linked to this rule. |
|-------|-------------|--------------------------------|