
GoodData SDK

Release 1.1.0

GoodData Corporation

Sep 08, 2022

CONTENTS:

- 1 Installation** **3**
- 1.1 Requirements 3
- 1.2 Installation 3
- 1.3 Troubleshooting 3

- 2 Services** **5**
- 2.1 Catalog Workspace Service 5
- 2.2 Catalog Workspace Content Service 10
- 2.3 Catalog Data Source Service 14
- 2.4 Catalog User Service 20
- 2.5 Catalog Permission Service 25
- 2.6 Catalog Organization Service 26
- 2.7 Insights Service 27
- 2.8 Compute Service 28
- 2.9 Table Service 28

- 3 API Reference** **31**
- 3.1 gooddata_sdk 31

- Python Module Index** **217**

- Index** **219**

GoodData Python SDK provides a clean and convenient Python API to interact with GoodData.CN and GoodData Cloud.

At the moment the SDK provides services to inspect and interact with the semantic layer and to consume analytics.

INSTALLATION

1.1 Requirements

- Python 3.7 or newer
- GoodData.CN or GoodData Cloud

1.2 Installation

Run the following command to install the `gooddata-sdk` package on your system:

```
pip install gooddata-sdk
```

1.3 Troubleshooting

- On MacOS, I am getting an error containig following message:

```
(Caused by SSLError(SSLCertVerificationError(1, '[SSL:
CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local
issuer certificate (_ssl.c:1129)'))).
```

This likely caused by Python and it occurs if you have installed Python installed directly from python.org. To mitigate this problem, please install your SSL certificates in *Macintosh HD -> Applications -> Python -> Install Certificates.command*.*.

SERVICES

All services are accessible by class `gooddata_sdk.GoodDataSdk`. The class forms an entry-point to the SDK.

To create an instance of `GoodDataSdk`:

```
from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# Now you can start calling services.
# For example, get a list of all workspaces from my GoodData.CN project
workspaces = sdk.catalog_workspace.list_workspaces()
```

2.1 Catalog Workspace Service

The `gooddata_sdk.catalog_workspace` service enables you to perform the following actions on workspaces:

- Get and list existing workspaces
- Update or delete existing workspaces
- Create new workspaces
- Store and restore workspaces from directory layout structure

The service supports two types of methods:

- Entity methods let you work with workspaces on a high level using simplified *CatalogWorkspace* entities.
- Declarative methods allow you to work with workspaces on a more granular level by fetching entire workspace layouts, including all of their nested objects.

2.1.1 Entity methods

The `gooddata_sdk.catalog_workspace` supports the following entity API calls:

- `create_or_update(workspace: CatalogWorkspace)`
Create a new workspace or overwrite an existing workspace with the same id.
- `get_workspace(workspace_id: str)`
Returns `CatalogWorkspace`.
Get an individual workspace.
- `delete_workspace(workspace_id: str)`
Delete a workspace with all its content - logical model and analytics model.
- `list_workspaces()`
Returns `List[CatalogWorkspace]`.
Get a list of all existing workspaces.

Example Usage

```
from gooddata_sdk import GoodDataSdk, CatalogWorkspace

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# List workspaces
workspaces = sdk.catalog_workspace.list_workspaces()

print(workspaces)
# [
#   CatalogWorkspace(id=demo, name=Demo),
#   CatalogWorkspace(id=demo_west, name=Demo West),
#   CatalogWorkspace(id=demo_west_california, name=Demo West California)
# ]

# Create new workspace entity locally
my_workspace_object = CatalogWorkspace(workspace_id="test_demo",
                                       name="Test demo",
                                       parent_id="demo")

# Create workspace
sdk.catalog_workspace.create_or_update(workspace=my_workspace_object)

# Edit local workspace entity
my_workspace_object.name = "Test"

# Update workspace
sdk.catalog_workspace.create_or_update(workspace=my_workspace_object)

# Get workspace
```

(continues on next page)

(continued from previous page)

```
workspace = sdk.catalog_workspace.get_workspace(workspace_id="test_demo")

print(workspace)
# CatalogWorkspace(id=test_demo, name=Test)

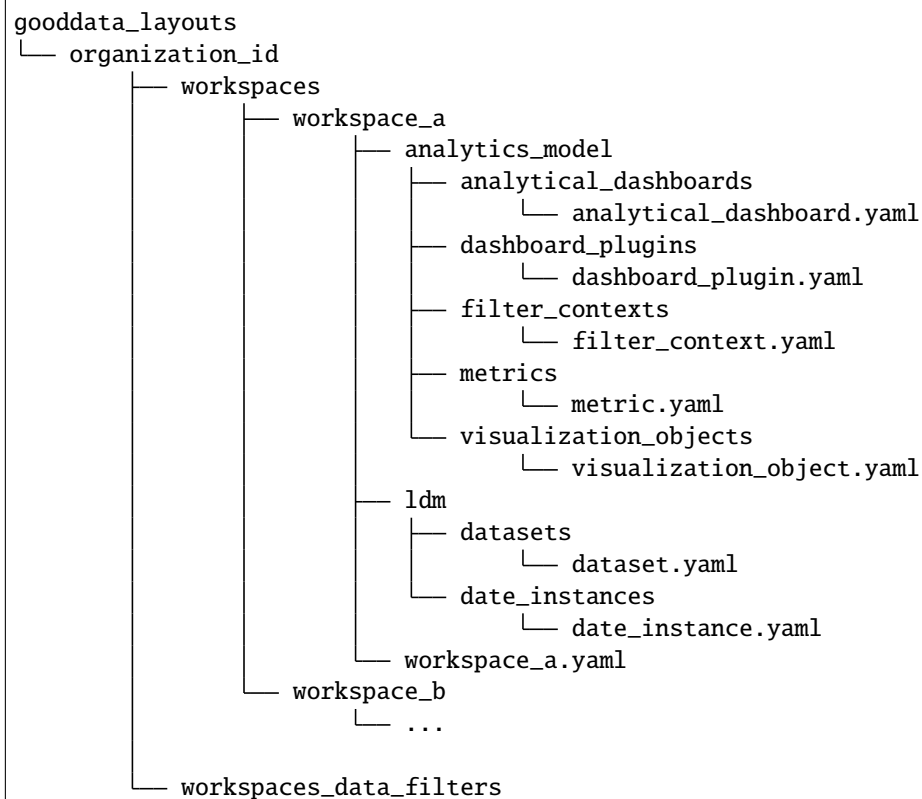
# Delete workspace
sdk.catalog_workspace.delete_workspace(workspace_id="test_demo")
```

2.1.2 Declarative methods

The `gooddata_sdk.catalog_workspace` supports the following declarative API calls:

Workspaces

- `get_declarative_workspaces()`
Returns *CatalogDeclarativeWorkspaces*.
Retrieve layout of all workspaces and their hierarchy.
- `put_declarative_workspaces(workspace: CatalogDeclarativeWorkspaces)`
Set layout of all workspaces and their hierarchy.
- `store_declarative_workspaces(layout_root_path: Path = Path.cwd())`
Store workspaces layouts in directory hierarchy.



(continues on next page)

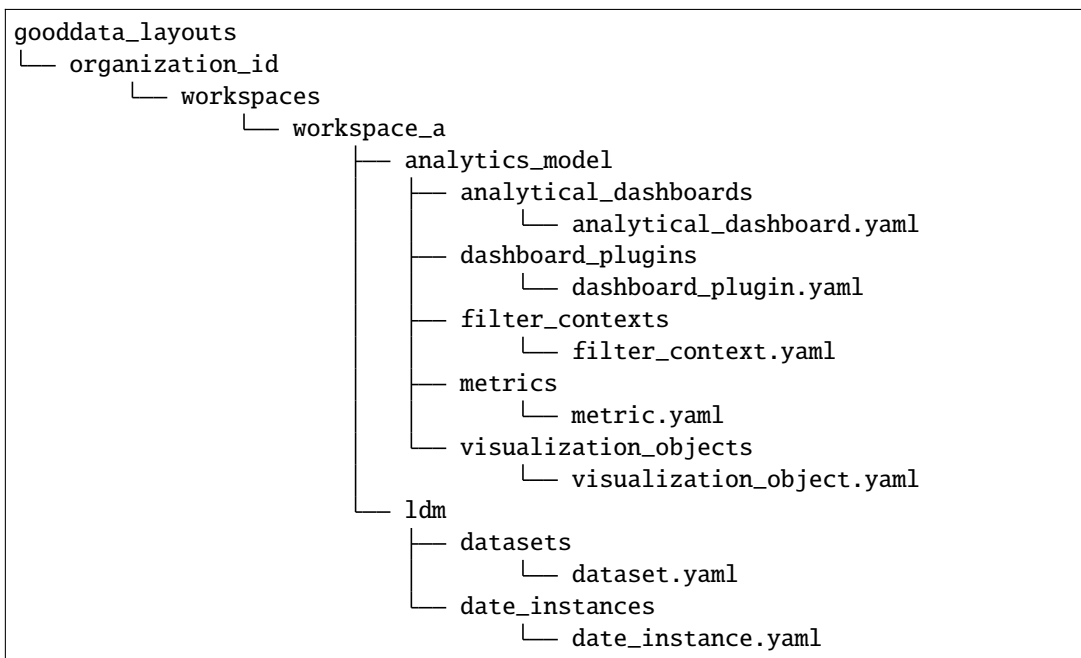
(continued from previous page)



- `load_declarative_workspaces(layout_root_path: Path = Path.cwd())`
Returns *CatalogDeclarativeWorkspaces*.
Load declarative workspaces layout, which was stored using *store_declarative_workspaces*.
- `load_and_put_declarative_workspaces(layout_root_path: Path = Path.cwd())`
This method combines *load_declarative_workspaces* and *put_declarative_workspaces* methods to load and set layouts stored using *store_declarative_workspaces*.

Workspace

- `get_declarative_workspace(workspace_id: str)`
Returns *CatalogDeclarativeWorkspaceModel*.
Retrieve a workspace layout.
- `put_declarative_workspace(workspace_id: str)`
Set a workspace layout.
- `store_declarative_workspace(workspace_id: str, layout_root_path: Path = Path.cwd())``
Store workspace layout in directory hierarchy.



- `load_declarative_workspace(workspace_id: str, layout_root_path: Path = Path.cwd())`
Returns *CatalogDeclarativeWorkspaceModel*.
Load declarative workspaces layout, which was stored using *store_declarative_workspace*.
- `load_and_put_declarative_workspace(workspace_id: str, layout_root_path: Path = Path.cwd())`

This method combines *load_declarative_workspace* and *put_declarative_workspace* methods to load and set layouts stored using *store_declarative_workspace*.

Workspace data filters

- `get_declarative_workspace_data_filters()`
Returns *CatalogDeclarativeWorkspaceDataFilters*.
Retrieve a workspace data filter layout.
- `put_declarative_workspace_data_filters(workspace_data_filters: CatalogDeclarativeWorkspaceDataFilters)`
Set a workspace data filter layout.
- `store_declarative_workspace_data_filters(layout_root_path: Path = Path.cwd())`
Store workspace data filters in directory hierarchy.

```

gooddata_layouts
├── organization_id
│   └── workspaces_data_filters
│       ├── filter_1.yaml
│       └── filter_2.yaml

```

- `load_declarative_workspace_data_filters(layout_root_path: Path = Path.cwd())`
Returns *CatalogDeclarativeWorkspaceDataFilters*.
Load declarative workspaces layout, which was stored using *store_declarative_workspace_data_filters*.
- `load_and_put_declarative_workspace_data_filters(layout_root_path: Path = Path.cwd())`
This method combines *load_declarative_workspace_data_filters* and *put_declarative_workspace_data_filters* methods to load and set layouts stored using *store_declarative_workspace_data_filters*.

Example Usage

```

from gooddata_sdk import GoodDataSdk
from pathlib import Path

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

backup_path = Path("workspace_hierarchy_backup")

# First create a backup of all workspace layout
sdk.catalog_workspace.store_declarative_workspaces(layout_root_path=backup_path)

# Get workspace layout
workspace_layout = sdk.catalog_workspace.get_declarative_workspace(workspace_id="demo")

```

(continues on next page)

```
# Modify workspace layout
workspace_layout.ldm.datasets[0].description = "This is test"

# Update the workspace layout on the server with your changes
sdk.catalog_workspace.put_declarative_workspace(workspace_id="demo",
                                                workspace=workspace_layout)

# If something goes wrong, use your backup to restore your workspaces from backup
sdk.catalog_workspace.load_and_put_declarative_workspaces(layout_root_path=backup_path)
```

2.2 Catalog Workspace Content Service

The `gooddata_sdk.catalog_workspace_content` service enables you to list catalog all objects from a workspace. These objects include:

- Datasets
- Metrics
- Facts
- Attributes

The service enables read, put, load and store of declarative layout for LDM (logical data model) and analytics model.

The service supports two types of methods:

- Entity methods let you work with workspace content on a high level using simplified entities.
- Declarative methods allow you to work with workspace content on a more granular level by fetching entire workspace content layouts, including all of their nested objects.

2.2.1 Entity methods

The `gooddata_sdk.catalog_workspace_content` supports the following entity API calls:

- `get_full_catalog(workspace_id: str)`
Returns *CatalogWorkspaceContent*.
Retrieve all datasets with attributes, facts, and metrics for a workspace.
- `get_attributes_catalog(workspace_id: str)`
Returns *list[CatalogAttribute]*
Retrieve all attributes for a workspace.
- `get_labels_catalog(workspace_id: str)`
Returns *list[CatalogLabel]*
Retrieve all labels for a workspace.
- `get_metrics_catalog(workspace_id: str)`
Returns *list[CatalogMetric]*
Retrieve all metrics for a workspace.

- `get_facts_catalog(workspace_id: str)`
Returns *list[CatalogFact]*
Retrieve all facts for a workspace.
- `get_dependent_entities_graph(workspace_id: str)`
Returns *CatalogDependentEntitiesResponse*

There are dependencies among all catalog objects, the chain is the following:
fact/attribute/label -> dataset -> metric -> insight -> dashboard

Some steps can be skipped, e.g. fact -> insight
We do not support table -> dataset dependency yet.

- `get_dependent_entities_graph_from_entry_points(workspace_id: str, dependent_entities_request: CatalogDependentEntitiesRequest)`
Returns *CatalogDependentEntitiesResponse*
Extends `get_dependent_entities_graph` with the entry point from which the graph is created.

Example Usage

```
from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

workspace_id = "demo"

# Read catalog for demo workspace
catalog = sdk.catalog_workspace_content.get_full_catalog(workspace_id)

# Print all dataset in the workspace
for dataset in catalog.datasets:
    print(str(dataset))

# Print all metrics in the workspace
for metric in catalog.metrics:
    print(str(metric))

# Read list of attributes for demo workspace
attributes = sdk.catalog_workspace_content.get_attributes_catalog(workspace_id)

# Read list of facts for demo workspace
facts = sdk.catalog_workspace_content.get_facts_catalog(workspace_id)
```

2.2.2 Declarative methods

The `gooddata_sdk.catalog_workspace_content` supports the following declarative API calls:

Logical data model (LDM)

- `get_declarative_ldm(workspace_id: str)`
Returns *CatalogDeclarativeModel*.
Retrieve a logical model layout. On *CatalogDeclarativeModel* user can call `modify_mapped_data_source(data_source_mapping: dict)` method, which substitutes data source id in datasets.
- `put_declarative_ldm(workspace_id: str, ldm: CatalogDeclarativeModel, validator: Optional[DataSourceValidator])`
Put a logical data model into a given workspace. You can pass an additional validator parameter which checks that for every data source id in the logical data model the corresponding data source exists.
- `store_declarative_ldm(workspace_id: str, layout_root_path: Path = Path.cwd())`
Store logical data model layout in directory hierarchy.

```

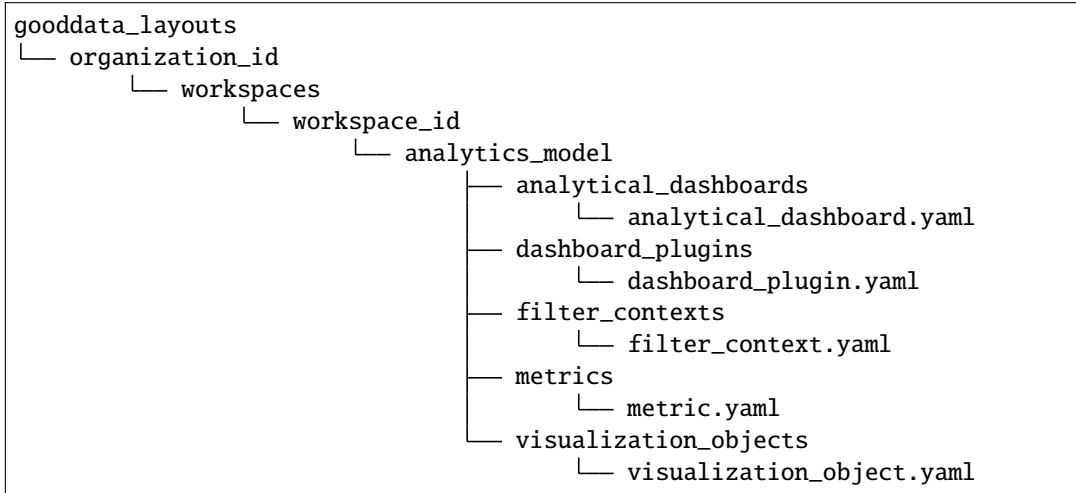
gooddata_layouts
├── organization_id
│   └── workspaces
│       └── workspace_id
│           └── analytics_model
│               └── ldm
│                   ├── datasets
│                   │   └── dataset.yaml
│                   └── date_instances
│                       └── date_instance.yaml

```

- `load_declarative_ldm(workspace_id: str, layout_root_path: Path = Path.cwd())`
Returns *CatalogDeclarativeModel*.
Load declarative LDM layout, which was stored using `store_declarative_ldm`.
- `load_and_put_declarative_ldm(workspace_id: str, layout_root_path: Path = Path.cwd(), validator: Optional[DataSourceValidator])`
This method combines `load_declarative_ldm` and `put_declarative_ldm` methods to load and set layouts stored using `store_declarative_ldm`. You can pass an additional validator parameter which checks that for every data source id in the logical data model the corresponding data source exists.

Analytics Model

- `get_declarative_analytics_model(workspace_id: str)`
Returns *CatalogDeclarativeAnalytics*.
Retrieve an analytics model layout.
- `put_declarative_analytics_model(workspace_id: str, analytics_model: CatalogDeclarativeAnalytics)`
Put an analytics model into a given workspace.
- `store_declarative_analytics_model(workspace_id: str, layout_root_path: Path = Path.cwd())`
Store declarative analytics model layout in directory hierarchy.



- `load_declarative_analytics_model(workspace_id: str, layout_root_path: Path = Path.cwd())`
Returns *CatalogDeclarativeAnalytics*.
Load declarative LDM layout, which was stored using *store_declarative_analytics_model*.
- `load_and_put_declarative_analytics_model(workspace_id: str, layout_root_path: Path = Path.cwd())`
This method combines *load_declarative_analytics_model* and *put_declarative_analytics_model* methods to load and set layouts stored using *store_declarative_analytics_model*.

Example usage:

```

from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

workspace_id = "demo"

```

(continues on next page)

```

# Get ldm object afterward you can modify it
ldm = sdk.catalog_workspace_content.get_declarative_ldm(workspace_id=workspace_id)

# Modify data source id for datasets
ldm.modify_mapped_data_source({"demo-test-ds": "demo-prod-ds"})

# Put ldm object back to server
sdk.catalog_workspace_content.put_declarative_ldm(workspace_id=workspace_id, ldm=ldm)

# Get analytics model object afterward you can modify it
analytics_model = sdk.catalog_workspace_content.get_declarative_analytics_
↳model(workspace_id=workspace_id)

# Put analytics model object back to server
sdk.catalog_workspace_content.put_declarative_analytics_model(workspace_id=workspace_id,
analytics_model=analytics_
↳model)

```

2.3 Catalog Data Source Service

The `gooddata_sdk.catalog_data_source` service enables you to manage data sources and list their tables. Data source object represents your database, which you integrate with GoodData.CN.

Generally there are two ways how to register data sources:

- The default way works for all data source types: You specify jdbc url, data source type and relevant credentials.
- Customized way for each of the different data source types. You specify custom attributes relevant for your data source and data source type and the url is set in background.

The service supports three types of methods:

- Entity methods let you work with data sources on a high level using simplified *CatalogDataSource* entities.
- Declarative methods allow you to work with data sources on a more granular level by fetching entire workspace layouts, including all of their nested objects.
- Action methods let you perform an execution of some form of computation.

2.3.1 Entity methods

The `gooddata_sdk.catalog_data_source` supports the following entity API calls:

- `create_or_update_data_source(data_source: CatalogDataSource)`
Create or update data source.
- `get_data_source(data_source_id: str)`
Returns *CatalogDataSource*.
Retrieve data source using data source id.
- `delete_data_source(data_source_id: str)`
Delete data source using data source id.

- `patch_data_source_attributes(data_source_id: str, attributes: dict)`
Allows you to apply changes to the given data source.
- `list_data_sources()`
Returns *List[CatalogDataSource]*.
Lists all data sources.
- `list_data_source_tables(data_source_id: str)`
Returns *List[CatalogDataSourceTable]*
Lists all tables for a data source specified by id.

Example Usage

```

from gooddata_sdk import GoodDataSdk
from gooddata_sdk import (
    CatalogDataSource,
    BasicCredentials,
    CatalogDataSourcePostgres,
    PostgresAttributes,
    CatalogDataSourceSnowflake,
    SnowflakeAttributes,
    CatalogDataSourceBigQuery,
    BigQueryAttributes,
    TokenCredentialsFromFile
)

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# Create (or update) data source using general interface - can be used for any type of
↳data source
# If data source already exists, it is updated
sdk.catalog_data_source.create_or_update_data_source(
    CatalogDataSource(
        id="test",
        name="Test2",
        data_source_type="POSTGRESQL",
        url="jdbc:postgresql://localhost:5432/demo",
        schema="demo",
        credentials=BasicCredentials(
            username="demouser",
            password="demopass",
        ),
        enable_caching=False,
        url_params=[("param", "value")]
    )
)

# Use Postgres specific interface

```

(continues on next page)

(continued from previous page)

```
sdk.catalog_data_source.create_or_update_data_source(  
    CatalogDataSourcePostgres(  
        id="test",  
        name="Test2",  
        db_specific_attributes=PostgresAttributes(  
            host="localhost", db_name="demo"  
        ),  
        schema="demo",  
        credentials=BasicCredentials(  
            username="demouser",  
            password="demopass",  
        ),  
        enable_caching=False,  
        url_params=[("param", "value")]  
    )  
)  
  
# Create Snowflake data source using specialized interface  
sdk.catalog_data_source.create_or_update_data_source(  
    CatalogDataSourceSnowflake(  
        id="test",  
        name="Test2",  
        db_specific_attributes=SnowflakeAttributes(  
            account="mycompany", warehouse="MYWAREHOUSE", db_name="MYDATABASE"  
        ),  
        schema="demo",  
        credentials=BasicCredentials(  
            username="demouser",  
            password="demopass",  
        ),  
        enable_caching=False,  
        url_params=[("param", "value")]  
    )  
)  
  
# BigQuery requires path to credentials file, where service account definition is stored  
sdk.catalog_data_source.create_or_update_data_source(  
    CatalogDataSourceBigQuery(  
        id="test",  
        name="Test",  
        db_specific_attributes=BigQueryAttributes(  
            project_id="project_id"  
        ),  
        schema="demo",  
        credentials=TokenCredentialsFromFile(  
            file_path=Path("credentials") / "bigquery_service_account.json"  
        ),  
        enable_caching=True,  
        cache_path=["cache_schema"],  
        url_params=[("param", "value")]  
    )  
)  
)
```

(continues on next page)

(continued from previous page)

```

# Look for other CatalogDataSource classes to find your data source type

# List data sources
data_sources = sdk.catalog_data_source.list_data_sources()

# Get single data source
data_sources = sdk.catalog_data_source.get_data_source(data_source_id='test')

# Patch data source attribute(s)
sdk.catalog_data_source.patch_data_source_attributes(data_source_id="test",
                                                    attributes={"name": "Name2"})

# Delete data source
sdk.catalog_data_source.delete_data_source(data_source_id='test')

```

2.3.2 Declarative methods

The `gooddata_sdk.catalog_data_source` supports the following declarative API calls:

Data sources

- `get_declarative_data_sources()`
Returns *CatalogDeclarativeDataSources*.
Retrieve all data sources, including their related physical model.
- `put_declarative_data_sources(declarative_data_sources: CatalogDeclarativeDataSources, credentials_path: Optional[Path] = None, test_data_sources: bool = False)`
Set all data sources, including their related physical model.
- `store_declarative_data_sources(layout_root_path: Path = Path.cwd())`
Store data sources layouts in directory hierarchy.

```

gooddata_layouts
├── organization_id
│   └── data_sources
│       ├── data_source_a
│       │   ├── pdm
│       │   │   ├── table_A.yaml
│       │   │   └── table_B.yaml
│       │   └── data_source_a.yaml
│       └── data_source_b
│           ├── pdm
│           │   ├── table_X.yaml
│           │   └── table_Y.yaml
│           └── data_source_b.yaml

```

- `load_declarative_data_sources(layout_root_path: Path = Path.cwd())`

Returns *CatalogDeclarativeDataSources*.

Load declarative data sources layout, which was stored using *store_declarative_data_sources*.

- `load_and_put_declarative_data_sources(layout_root_path: Path = Path.cwd(), credentials_path: Optional[Path] = None, test_data_sources: bool = False)`

This method combines *load_declarative_data_sources* and *put_declarative_data_sources* methods to load and set layouts stored using *store_declarative_data_sources*.

Physical data model (PDM)

- `get_declarative_pdm(data_source_id: str)`

Returns *CatalogDeclarativeTables*.

Retrieve physical model for a given data source.

- `put_declarative_pdm(data_source_id: str, declarative_tables: CatalogDeclarativeTables)`

Set physical model for a given data source.

- `store_declarative_pdm(data_source_id: str, layout_root_path: Path = Path.cwd())`

Store physical model layout in directory hierarchy for a given data source.

```

gooddata_layouts
├── organization_id
│   └── data_sources
│       └── data_source_a
│           └── pdm
│               ├── table_A.yaml
│               └── table_B.yaml
    
```

- `load_declarative_pdm(data_source_id: str, layout_root_path: Path = Path.cwd())`

Returns *CatalogDeclarativeTables*.

Load declarative physical model layout, which was stored using *store_declarative_pdm* for a given data source.

- `load_and_put_declarative_pdm(self, data_source_id: str, layout_root_path: Path = Path.cwd())`

This method combines *load_declarative_pdm* and *put_declarative_pdm* methods to load and set layouts stored using *store_declarative_pdm*.

Example usage:

```

from gooddata_sdk import GoodDataSdk
from pathlib import Path

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# Get all data sources
    
```

(continues on next page)

(continued from previous page)

```

ds_objects = sdk.catalog_data_source.get_declarative_data_sources()

print(ds_objects.data_sources[0])
# CatalogDeclarativeDataSource(id=demo-test-ds, type=POSTGRESQL)

# Put data sources with credentials and test data source connection before put
sdk.catalog_data_source.put_declarative_data_sources(declarative_data_sources=ds_objects,
                                                    credentials_path=Path("credentials"),
                                                    test_data_sources=True)

```

2.3.3 Action methods

The `gooddata_sdk.catalog_data_source` supports the following action API calls:

- `generate_logical_model(data_source_id: str, generate_ldm_request: CatalogGenerateLdmRequest)`
Returns *CatalogDeclarativeModel*.
Generate logical data model for a data source.
- `register_upload_notification(data_source_id: str)`
Invalidate cache of your computed reports to force your analytics to be recomputed.
- `scan_data_source(data_source_id: str, scan_request: CatalogScanModelRequest = CatalogScanModelRequest(), report_warnings: bool = False)`
Returns *CatalogScanResultPdm*.
Scan data source specified by its id and optionally by specified scan request. *CatalogScanResultPdm* contains PDM and warnings. Warnings contain information about columns which were not added to the PDM because their data types are not supported. Additional parameter `report_warnings` can be passed to suppress or to report warnings. By default warnings are returned but not reported to STDOUT. If you set `report_warnings` to `True`, warnings are reported to STDOUT.
- `scan_and_put_pdm(data_source_id: str, scan_request: CatalogScanModelRequest = CatalogScanModelRequest())`
This method combines `scan_data_source` and `put_declarative_pdm` methods.
- `scan_schemata(data_source_id: str)`
Returns *list[str]*.
Returns a list of schemas that exist in the database and can be configured in the data source entity. Data source managers like Dremio or Drill can work with multiple schemas and schema names can be injected into `scan_request` to filter out tables stored in the different schemas.
- `test_data_sources_connection(declarative_data_sources: CatalogDeclarativeDataSources, credentials_path: Optional[Path] = None)`
Tests connection to declarative data sources. If `credentials_path` is omitted then the connection is tested with empty credentials. In case some connection failed the `ValueError` is raised with information about why the connection to the data source failed, e.g. `host unreachable or invalid login or password`".

Example of credentials YAML file:

```
::
```

data_sources:

demo-test-ds: "demopass" demo-bigquery-ds: "~/home/secrets.json"

Example usage:

```
from gooddata_sdk import GoodDataSdk, CatalogGenerateLdmRequest

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

data_source_id = "demo-test-ds"

# Scan schemata of the data source
schemata = sdk.catalog_data_source.scan_schemata(data_source_id=data_source_id)
print(schemata)
# ['demo']

# Scan and put pdm
sdk.catalog_data_source.scan_and_put_pdm(data_source_id=data_source_id)

# Define request for generating ldm
generate_ldm_request = CatalogGenerateLdmRequest(separator="__")

# Generate ldm
declarative_model = sdk.catalog_data_source.generate_logical_model(data_source_id=data_
↪source_id,
                                                                    generate_ldm_
↪request=generate_ldm_request)

# Invalidate cache of your computed reports
sdk.catalog_data_source.register_upload_notification(data_source_id=data_source_id)
```

2.4 Catalog User Service

The `gooddata_sdk.catalog_user` service enables you to perform the following actions on users and user groups:

- Get and list existing users and user groups
- Update or delete existing users and user groups
- Create new users and user groups
- Store and restore users and user groups from directory layout structure

The service supports two types of methods:

- Entity methods let you work with users and user groups on a high level using simplified *CatalogUser* and *CatalogUserGroup* entities.
- Declarative methods allow you to work with users and user groups on a more granular level by fetching entire users and user groups layouts.

2.4.1 Entity methods

Users

The `gooddata_sdk.catalog_user` supports the following user entity API calls:

- `create_or_update_user(user: CatalogUser)`
Create a new user or overwrite an existing user.
- `get_user(user_id: str)`
Returns `CatalogUser`.
Get an individual user.
- `delete_user(user_id: str)`
Delete a user.
- `list_users()`
Returns `List[CatalogUser]`.
Get a list of all existing users.

Example Usage

```

from gooddata_sdk import GoodDataSdk, CatalogUser

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# List users
users = sdk.catalog_user.list_users()

print(users)
# [
#   CatalogUser(id='demo2',
#               attributes=CatalogUserAttributes(authentication_id=
# → 'CiRmYmNhNDkwOS04YzYxLTRmMTYtODI3NC1iNzI0Njk1Y2FmNTESBWxvY2Fs'),
#               relationships=CatalogUserRelationships(user_
# → groups=CatalogUserGroupsData(data=[CatalogUserGroup(id='demoGroup',
# → relationships=None)])),
#   ...
# ]

# Define user
user = CatalogUser.init(user_id="abc", authentication_id="xyz", user_group_ids=["demoGroup
→"])

# Create user
sdk.catalog_user.create_or_update_user(user=user)

# Delete user
sdk.catalog_user.delete_user(user_id=user.id)

```

User groups

The `gooddata_sdk.catalog_user` supports the following user groups entity API calls:

- `create_or_update_user_group(user_group: CatalogUserGroup)`
Create a new user group or overwrite an existing user group.
- `get_user_group(user_group_id: str)`
Returns `CatalogUserGroup`.
Get an individual user group.
- `delete_user_group(user_group_id: str)`
Delete a user group.
- `list_user_groups()`
Returns `List[CatalogUserGroup]`.
Get a list of all existing user groups.

Example Usage

```
from gooddata_sdk import GoodDataSdk, CatalogUserGroup

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# List user groups
user_groups = sdk.catalog_user.list_user_groups()

print(user_groups)
#[
#   CatalogUserGroup(id='adminGroup', relationships=None),
#   CatalogUserGroup(id='adminQA1Group',
#   ↪relationships=CatalogUserGroupRelationships(parents=CatalogUserGroupParents(data=[CatalogUserGroup(id
#   ↪'adminGroup', relationships=None])))
#   ...
#]

# Define user
user_group = CatalogUserGroup.init(user_group_id="xyz", user_group_parent_ids=["demoGroup
↪"])

# Create user
sdk.catalog_user.create_or_update_user_group(user_group=user_group)

# Delete user
sdk.catalog_user.delete_user_group(user_group_id=user_group.id)
```

2.4.2 Declarative methods

Users

The `gooddata_sdk.catalog_user` supports the following declarative user API calls:

- `get_declarative_users()`
Returns `CatalogDeclarativeUsers`.
Retrieve all users including authentication properties.
- `put_declarative_users(users: CatalogDeclarativeUsers)`
Set all users and their authentication properties.
- `store_declarative_users(layout_root_path: Path = Path.cwd())`
Store users in directory hierarchy.

```
gooddata_layouts
├── organization_id
│   └── users
│       └── users.yaml
```

- `load_declarative_users(layout_root_path: Path = Path.cwd())`
Load users from directory hierarchy.
- `load_and_put_declarative_users(layout_root_path: Path = Path.cwd())`
This method combines `load_declarative_users` and `put_declarative_users` methods to load and set users stored using `store_declarative_users`.

Example Usage

```
from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# Get user layout
user_layout = sdk.catalog_user.get_declarative_users()

print(user_layout)
# CatalogDeclarativeUsers(
#     users=[
#         CatalogDeclarativeUser(id='admin',
#                                 auth_id=None,
#                                 user_groups=[CatalogUserGroupIdentifier(id=
# ↪ 'adminGroup', type='userGroup')]),
#         CatalogDeclarativeUser(id='demo',...
#     ]
# )
# ...

# Modify user layout
user_layout.users = []
```

(continues on next page)

```
# Update user layout
sdk.catalog_user.put_declarative_users(users=user_layout)
```

User groups

The `gooddata_sdk.catalog_user` supports the following declarative user groups API calls:

- `get_declarative_user_groups()`
Returns *CatalogDeclarativeUserGroups*.
Retrieve all user-groups eventually with parent group.
- `put_declarative_user_groups(user_groups: CatalogDeclarativeUserGroups)`
Set all user groups with their parents eventually.
- `store_declarative_user_groups(layout_root_path: Path = Path.cwd())`
Store user groups in directory hierarchy.

```
gooddata_layouts
├── organization_id
│   └── user_groups
│       └── user_groups.yaml
```

- `load_declarative_user_groups(layout_root_path: Path = Path.cwd())`
Returns *CatalogDeclarativeUserGroups*.
Load user groups from directory hierarchy.
- `load_and_put_declarative_user_groups(layout_root_path: Path = Path.cwd())`
This method combines *load_declarative_user_groups* and *put_declarative_user_groups* methods to load and set user groups stored using *store_declarative_user_groups*.

Example Usage

```
from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# Get user layout
user_group_layout = sdk.catalog_user.get_declarative_user_groups()

print(user_group_layout)
# CatalogDeclarativeUserGroups(
#     user_groups=[
#         CatalogDeclarativeUserGroup(id='adminGroup', parents=None),
#     ]
# )
```

(continues on next page)

(continued from previous page)

```
# Modify user group layout
user_group_layout.user_groups = []

# Update user group layout
sdk.catalog_user.put_declarative_users(users=user_group_layout)
```

Users and user groups

The `gooddata_sdk.catalog_user` supports the following declarative users and user groups API calls:

- `get_declarative_users_user_groups()`
Returns *CatalogDeclarativeUsersUserGroups*.
Retrieve all users and all user-groups.
- `put_declarative_users_user_groups(users_user_groups: CatalogDeclarativeUsersUserGroups)`
Set all users and user groups.
- `store_declarative_users_user_groups(layout_root_path: Path = Path.cwd())`
Store users and user groups in directory hierarchy.

```
gooddata_layouts
├── organization_id
│   ├── users
│   │   └── users.yaml
│   └── user_groups
│       └── user_groups.yaml
```

- `load_declarative_users_user_groups(layout_root_path: Path = Path.cwd())`
Returns *CatalogDeclarativeUsersUserGroups*.
Load users and user groups from directory hierarchy.
- `load_and_put_declarative_users_user_groups(layout_root_path: Path = Path.cwd())`
This method combines *load_declarative_users_user_groups* and *put_declarative_users_user_groups* methods to load and set users and user groups stored using *store_declarative_users_user_groups*.

2.5 Catalog Permission Service

The `gooddata_sdk.catalog_permission` service enables you to perform the following actions on permissions:

- Get and set declarative permissions

2.5.1 Declarative methods

The `gooddata_sdk.catalog_permission` supports the following declarative API calls:

- `get_declarative_permissions(workspace_id: str)`
Returns `CatalogDeclarativeWorkspacePermissions`.
Retrieve current set of permissions of the workspace in a declarative form.
- `put_declarative_permissions(workspace_id: str, declarative_workspace_permissions: CatalogDeclarativeWorkspacePermissions)`
Set effective permissions for the workspace.

Example Usage

```
from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

workspace_id = "demo"

# Get permissions in declarative form
declarative_permissions = sdk.catalog_permission.get_declarative_permissions(workspace_
↳id=workspace_id)

declarative_permissions.permissions = []

# Update permissions on the server with your changes
sdk.catalog_permission.put_declarative_permissions(workspace_id=workspace_id,
                                                    declarative_workspace_
↳permissions=declarative_permissions)
```

2.6 Catalog Organization Service

The `gooddata_sdk.catalog_organization` service enables you to perform the following actions on organization:

- Update OIDC parameters
- Update organization name

2.6.1 Entity methods

The `gooddata_sdk.catalog_organization` supports the following entity API calls:

- `update_oidc_parameters(oauth_issuer_location: Optional[str] = None, oauth_client_id: Optional[str] = None, oauth_client_secret: Optional[str] = None)`
Update OIDC parameters of organization.
- `update_name(name: str)`
Update name of organization.

Example Usage

```
from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

# Update organization name
sdk.catalog_organization.update_name(name="new_organization_name")

# Update OIDC provider
sdk.catalog_organization.update_oidc_parameters(oauth_client_id="oauth_client_id",
                                                oauth_issuer_location="oauth_issuer_
↪location",
                                                oauth_client_secret="oauth_client_secret
↪")
```

2.7 Insights Service

The `gooddata_sdk.insights` service gives you access to insights stored in a workspace. It can retrieve all the insights from a workspace or one insight based on its name. Insight instance is the input for other services like a Table service

2.7.1 Entity methods

The `gooddata_sdk.insights` supports the following entity API calls:

- `get_insights(workspace_id: str)`
Returns *list[Insight]*.
Retrieve a list of Insight objects.

Example usage:

Read all insights in a workspace:

```
from gooddata_sdk import GoodDataSdk
```

(continues on next page)

(continued from previous page)

```
# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

workspace_id = "demo"

# Reads insights from workspace
insights = sdk.insights.get_insights(workspace_id)
# Print all fetched insights
for insight in insights:
    print(str(insight))
```

2.8 Compute Service

The `gooddata_sdk.compute` service drives computation of analytics for GoodData.CN workspaces. The prescription of what to compute is encapsulated by the `ExecutionDefinition` which consists of attributes, metrics, filters and definition of dimensions that influence how to organize the data in the result.

Higher level services like Table service use Compute service to execute computation in GoodData.CN. Higher level service is also responsible for results presentation to the user e.g. in tabular form.

2.8.1 Entity methods

The `gooddata_sdk.compute` supports the following entity API calls:

- `for_exec_def(workspace_id: str, exec_def: ExecutionDefinition)`
Returns *ExecutionResponse*.
Starts computation in GoodData.CN workspace, using the provided execution definition.

2.9 Table Service

The `gooddata_sdk.table` service allows you to consume analytics in typical tabular format. The service allows free-form computations and computations of data for GoodData.CN Insights.

2.9.1 Entity methods

The `gooddata_sdk.table` supports the following entity API calls:

- `for_insight(workspace_id: str, insight: Insight)`
Returns *ExecutionTable*.
Retrieve data as an `ExecutionTable` from the given insight.
- `for_items(workspace_id: str, items: list[Union[Attribute, Metric]], filters: Optional[list[Filter]] = None)`

Returns *ExecutionTable*.

Retrieve data as an *ExecutionTable* from the given list of attributes/metrics, and filters.

Example usage:

Get tabular data for an insight defined on your GoodData.CN server:

```

from gooddata_sdk import GoodDataSdk

# GoodData.CN host in the form of uri eg. "http://localhost:3000"
host = "http://localhost:3000"
# GoodData.CN user token
token = "some_user_token"
sdk = GoodDataSdk.create(host, token)

workspace_id = "demo"
insight_id = "some_insight_id_in_demo_workspace"

# Reads insight from workspace
insight = sdk.insights.get_insight(workspace_id, insight_id)

# Triggers computation for the insight. the result will be returned in a tabular form
table = sdk.tables.for_insight(workspace_id, insight)

# This is how you can read data row-by-row and do something with it
for row in table.read_all():
    print(row)

# An example of data printed for insight top_10_products
# {'781952e728204dcf923142910cc22ae2': 'Biolid', 'fe513cef1c6244a5ac21c5f49c56b108': 'Outdoor
↪', '77dc71bbac92412bac5f94284a5919df': 34697.71}
# {'781952e728204dcf923142910cc22ae2': 'ChalkTalk', 'fe513cef1c6244a5ac21c5f49c56b108': 'Home
↪', '77dc71bbac92412bac5f94284a5919df': 17657.35}
# {'781952e728204dcf923142910cc22ae2': 'Elentrix', 'fe513cef1c6244a5ac21c5f49c56b108':
↪'Outdoor', '77dc71bbac92412bac5f94284a5919df': 27662.09}
# {'781952e728204dcf923142910cc22ae2': 'Integres', 'fe513cef1c6244a5ac21c5f49c56b108':
↪'Outdoor', '77dc71bbac92412bac5f94284a5919df': 47766.74}
# {'781952e728204dcf923142910cc22ae2': 'Magnemo', 'fe513cef1c6244a5ac21c5f49c56b108':
↪'Electronics', '77dc71bbac92412bac5f94284a5919df': 44026.52}
# {'781952e728204dcf923142910cc22ae2': 'Neptide', 'fe513cef1c6244a5ac21c5f49c56b108': 'Outdoor
↪', '77dc71bbac92412bac5f94284a5919df': 99440.44}
# {'781952e728204dcf923142910cc22ae2': 'Optique', 'fe513cef1c6244a5ac21c5f49c56b108': 'Home',
↪'77dc71bbac92412bac5f94284a5919df': 40307.76}
# {'781952e728204dcf923142910cc22ae2': 'PortaCode', 'fe513cef1c6244a5ac21c5f49c56b108':
↪'Electronics', '77dc71bbac92412bac5f94284a5919df': 18841.17}
# {'781952e728204dcf923142910cc22ae2': 'Slacks', 'fe513cef1c6244a5ac21c5f49c56b108': 'Clothing
↪', '77dc71bbac92412bac5f94284a5919df': 18469.15}
# {'781952e728204dcf923142910cc22ae2': 'T-Shirt', 'fe513cef1c6244a5ac21c5f49c56b108':
↪'Clothing', '77dc71bbac92412bac5f94284a5919df': 17937.49}

```


API REFERENCE

gooddata_sdk

The *gooddata-sdk* package aims to provide clean and convenient Python APIs to interact with GoodData.CN.

3.1 gooddata_sdk

The *gooddata-sdk* package aims to provide clean and convenient Python APIs to interact with GoodData.CN.

At the moment the SDK provides services to inspect and interact with the Semantic Model and consume analytics.

Modules

gooddata_sdk.catalog

gooddata_sdk.client

Module containing a class that provides access to meta-data and afm services.

gooddata_sdk.compute

gooddata_sdk.insight

gooddata_sdk.sdk

gooddata_sdk.support

gooddata_sdk.table

gooddata_sdk.type_converter

gooddata_sdk.utils

3.1.1 gooddata_sdk.catalog

Modules

gooddata_sdk.catalog.base

gooddata_sdk.catalog.catalog_service_base

gooddata_sdk.catalog.data_source

gooddata_sdk.catalog.entity

gooddata_sdk.catalog.identifier

gooddata_sdk.catalog.organization

gooddata_sdk.catalog.permission

gooddata_sdk.catalog.setting

gooddata_sdk.catalog.types

gooddata_sdk.catalog.user

gooddata_sdk.catalog.workspace

gooddata_sdk.catalog.base

Functions

value_in_allowed(instance, attribute, value)

gooddata_sdk.catalog.base.value_in_allowed

`gooddata_sdk.catalog.base.value_in_allowed`(*instance: Type[Base], attribute: Attribute, value: str*) → None

Classes

Base()

gooddata_sdk.catalog.base.Base

class `gooddata_sdk.catalog.base.Base`

Bases: object

`__init__`() → None

Method generated by attrs for class Base.

Methods

<code>__init__</code> ()	Method generated by attrs for class Base.
<code>client_class</code> ()	
<code>from_api</code> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (data[, camel_case])	Creates object from dictionary.
<code>to_api</code> ()	
<code>to_dict</code> ([camel_case])	Converts object into dictionary.

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.catalog_service_base

Classes

CatalogServiceBase(api_client)

gooddata_sdk.catalog.catalog_service_base.CatalogServiceBase

class gooddata_sdk.catalog.catalog_service_base.CatalogServiceBase(*api_client*:
GoodDataApiClient)

Bases: object

__init__(*api_client*: GoodDataApiClient) → None

Methods

__init__(api_client)

get_organization()

layout_organization_folder(layout_root_path)

Attributes

organization_id

gooddata_sdk.catalog.data_source

Modules

gooddata_sdk.catalog.data_source.action_requests

gooddata_sdk.catalog.data_source.declarative_model

gooddata_sdk.catalog.data_source.entity_model

gooddata_sdk.catalog.data_source.service

gooddata_sdk.catalog.data_source.validation

gooddata_sdk.catalog.data_source.action_requests

Modules

gooddata_sdk.catalog.data_source.

action_requests.ldm_request

gooddata_sdk.catalog.data_source.

action_requests.scan_model_request

gooddata_sdk.catalog.data_source.action_requests.ldm_request

Classes

CatalogGenerateLdmRequest(*[, separator, ...])

gooddata_sdk.catalog.data_source.action_requests.ldm_request.CatalogGenerateLdmRequest

```
class gooddata_sdk.catalog.data_source.action_requests.ldm_request.CatalogGenerateLdmRequest(*,
    sep: str = '_',
    generate_long_ids: Optional[bool] = None,
    table_prefix: Optional[str] = None,
    view_prefix: Optional[str] = None,
    primary_label_prefix: Optional[str] = None,
    secondary_label_prefix: Optional[str] = None,
    fact_prefix: Optional[str] = None,
    date_granularity_prefix: Optional[str] = None,
    grain_prefix: Optional[str] = None,
    reference_prefix: Optional[str] = None,
```


Bases: *Base*

```
__init__(*, separator: str = '_', generate_long_ids: Optional[bool] = None, table_prefix: Optional[str] = None, view_prefix: Optional[str] = None, primary_label_prefix: Optional[str] = None, secondary_label_prefix: Optional[str] = None, fact_prefix: Optional[str] = None, date_granularities: Optional[str] = None, grain_prefix: Optional[str] = None, reference_prefix: Optional[str] = None, grain_reference_prefix: Optional[str] = None, denorm_prefix: Optional[str] = None, wdf_prefix: Optional[str] = None) → None
```

Method generated by attrs for class `CatalogGenerateLdmRequest`.

Methods

<code><i>__init__</i>(*[, separator, generate_long_ids, ...])</code>	Method generated by attrs for class <code>CatalogGenerateLdmRequest</code> .
<code><i>client_class</i>()</code>	
<code><i>from_api</i>(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code><i>from_dict</i>(data[, camel_case])</code>	Creates object from dictionary.
<code><i>to_api</i>()</code>	
<code><i>to_dict</i>([camel_case])</code>	Converts object into dictionary.

Attributes

separator

generate_long_ids

table_prefix

view_prefix

primary_label_prefix

secondary_label_prefix

fact_prefix

date_granularities

grain_prefix

reference_prefix

grain_reference_prefix

denorm_prefix

wdf_prefix

classmethod from_api(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(*data: Dict[str, Any]*, *camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.data_source.action_requests.scan_model_request

Functions

one_scan_true(instance, *args)

`gooddata_sdk.catalog.data_source.action_requests.scan_model_request.one_scan_true`

`gooddata_sdk.catalog.data_source.action_requests.scan_model_request.one_scan_true`(*instance:* CatalogScanModelRequest, **args:* Any) → None

Classes

`CatalogScanModelRequest`(*[, separator, ...])

`gooddata_sdk.catalog.data_source.action_requests.scan_model_request.CatalogScanModelRequest`

```
class gooddata_sdk.catalog.data_source.action_requests.scan_model_request.CatalogScanModelRequest(*,
                                                    separator: Optional[str] = None,
                                                    scan_tables: bool = True,
                                                    scan_views: bool = False,
                                                    table_prefix: Optional[str] = None,
                                                    view_prefix: Optional[str] = None) → None
```

Bases: `Base`

`__init__`(*, separator: str = '_', scan_tables: bool = True, scan_views: bool = False, table_prefix: Optional[str] = None, view_prefix: Optional[str] = None) → None

Method generated by attrs for class `CatalogScanModelRequest`.

Methods

<code>__init__(*[, separator, scan_tables, ...])</code>	Method generated by attrs for class <code>CatalogScan-ModelRequest</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>separator</code>
<code>scan_tables</code>
<code>scan_views</code>
<code>table_prefix</code>
<code>view_prefix</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.data_source.declarative_model`

Modules

<code>gooddata_sdk.catalog.data_source.declarative_model.data_source</code>
<code>gooddata_sdk.catalog.data_source.declarative_model.physical_model</code>

gooddata_sdk.catalog.data_source.declarative_model.data_source**Classes**

CatalogDeclarativeDataSource(*, id, type, ...)

CatalogDeclarativeDataSources(*, data_sources)

gooddata_sdk.catalog.data_source.declarative_model.data_source.CatalogDeclarativeDataSource

```

class gooddata_sdk.catalog.data_source.declarative_model.data_source.CatalogDeclarativeDataSource(*,
                                                    id:
                                                    str,
                                                    type:
                                                    str,
                                                    name:
                                                    str,
                                                    url:
                                                    str,
                                                    schema:
                                                    str,
                                                    enable_cache:
                                                    Optional[bool] =
                                                    None,
                                                    pdm:
                                                    Optional[bool] =
                                                    None,
                                                    cache_name:
                                                    Optional[str] =
                                                    None,
                                                    user_name:
                                                    Optional[str] =
                                                    None,
                                                    permissions:
                                                    List[CatalogDeclarativeDataSourcePermission] =
                                                    NOTHING)

```

Bases: *Base*

```
__init__(*, id: str, type: str, name: str, url: str, schema: str, enable_caching: Optional[bool] = None, pdm:
    Optional[CatalogDeclarativeTables] = None, cache_path: Optional[List[str]] = None, username:
    Optional[str] = None, permissions: List[CatalogDeclarativeDataSourcePermission] = NOTHING)
    → None
```

Method generated by attrs for class CatalogDeclarativeDataSource.

Methods

<code><i>__init__</i>(*, id, type, name, url, schema[, ...])</code>	Method generated by attrs for class CatalogDeclarativeDataSource.
<code>client_class()</code>	
<code>data_source_folder(data_sources_folder, ...)</code>	
<code><i>from_api</i>(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code><i>from_dict</i>(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(data_sources_folder, ...)</code>	
<code>store_to_disk(data_sources_folder)</code>	
<code>to_api([password, token, ...])</code>	
<code><i>to_dict</i>([camel_case])</code>	Converts object into dictionary.
<code>to_test_request([password, token])</code>	

Attributes

id

type

name

url

schema

enable_caching

pdm

cache_path

username

permissions

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.data_source.declarative_model.data_source.CatalogDeclarativeDataSources

class `gooddata_sdk.catalog.data_source.declarative_model.data_source.CatalogDeclarativeDataSources(*, data_sources: List[CatalogDeclarativeDataSource])`

Bases: *Base*

`__init__`(*, *data_sources: List[CatalogDeclarativeDataSource]*) → None

Method generated by attrs for class CatalogDeclarativeDataSources.

Methods

<code>__init__(*, data_sources)</code>	Method generated by attrs for class CatalogDeclarativeDataSources.
<code>client_class()</code>	
<code>data_sources_folder(layout_organization_folder)</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(layout_organization_folder)</code>	
<code>store_to_disk(layout_organization_folder)</code>	
<code>to_api([credentials])</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>data_sources</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.data_source.declarative_model.physical_model`

Modules

<code>gooddata_sdk.catalog.data_source.declarative_model.physical_model.column</code>
<code>gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm</code>
<code>gooddata_sdk.catalog.data_source.declarative_model.physical_model.table</code>

`gooddata_sdk.catalog.data_source.declarative_model.physical_model.column`

Classes

`CatalogDeclarativeColumn(*, name, data_type)`

`gooddata_sdk.catalog.data_source.declarative_model.physical_model.column.CatalogDeclarativeColumn`

`class gooddata_sdk.catalog.data_source.declarative_model.physical_model.column.CatalogDeclarativeColumn`

Bases: `Base`

`__init__(*, name: str, data_type: str, is_primary_key: Optional[bool] = None, referenced_table_id: Optional[str] = None, referenced_table_column: Optional[str] = None) → None`

Method generated by attrs for class `CatalogDeclarativeColumn`.

Methods

<code>__init__(*, name, data_type[, ...])</code>	Method generated by attrs for class CatalogDeclarativeColumn.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>name</code>
<code>data_type</code>
<code>is_primary_key</code>
<code>referenced_table_id</code>
<code>referenced_table_column</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm`

Functions

<code>get_pdm_folder(data_source_folder)</code>

gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm.get_pdm_folder

```
gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm.get_pdm_folder(data_source_folder:
                                                                                               Path)
                                                                                               →
                                                                                               Path
```

Classes

```
CatalogDeclarativeTables(*[, tables])
```

```
CatalogScanResultPdm(*[, pdm, warnings])
```

gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm.CatalogDeclarativeTables

```
class gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm.CatalogDeclarativeTables(*,
                                                                                               ta-
                                                                                               ble
                                                                                               Lis
                                                                                               =
                                                                                               NO
                                                                                               IN
```

Bases: *Base*

```
__init__(*, tables: List[CatalogDeclarativeTable] = NOTHING) → None
```

Method generated by attrs for class CatalogDeclarativeTables.

Methods

<code>__init__(*[, tables])</code>	Method generated by attrs for class CatalogDeclarativeTables.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(data_source_folder)</code>	
<code>store_to_disk(data_source_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

tables

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm.CatalogScanResultPdm

```
class gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm.CatalogScanResultPdm(*,
                                                                                               pdm:
                                                                                               Cat-
                                                                                               a-
                                                                                               logDecl
                                                                                               a-
                                                                                               tiveTa-
                                                                                               bles
                                                                                               =
                                                                                               Cat-
                                                                                               a-
                                                                                               logDecl
                                                                                               a-
                                                                                               tiveTa-
                                                                                               bles(tab
                                                                                               warn-
                                                                                               ings:
                                                                                               List[Dic
                                                                                               =
                                                                                               NOTH-
                                                                                               ING)
```

Bases: *Base*

```
__init__(*, pdm: CatalogDeclarativeTables = CatalogDeclarativeTables(tables=[]), warnings: List[Dict] =
        NOTHING) → None
```

Method generated by attrs for class CatalogScanResultPdm.

Methods

<code>__init__(*[, pdm, warnings])</code>	Method generated by attrs for class CatalogScanResultPdm.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>pdm</code>	
<code>warnings</code>	

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.data_source.declarative_model.physical_model.table`

Classes

<code>CatalogDeclarativeTable(*, id, type, path, ...)</code>
--

`gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.CatalogDeclarativeTable`

`class gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.CatalogDeclarativeTable(*`

Bases: *Base*

`__init__`(*, *id*: str, *type*: str, *path*: List[str], *columns*: List[CatalogDeclarativeColumn], *name_prefix*: Optional[str] = None) → None

Method generated by attrs for class CatalogDeclarativeTable.

Methods

<code>__init__</code> (*, <i>id</i> , <i>type</i> , <i>path</i> , <i>columns</i> [, ...])	Method generated by attrs for class CatalogDeclarativeTable.
<code>client_class</code> ()	
<code>from_api</code> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (data[, <i>camel_case</i>])	Creates object from dictionary.
<code>load_from_disk</code> (table_file_path)	
<code>store_to_disk</code> (pdm_folder)	
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

<code>id</code>
<code>type</code>
<code>path</code>
<code>columns</code>
<code>name_prefix</code>

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any]*, *camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.data_source.entity_model`

Modules

`gooddata_sdk.catalog.data_source.
entity_model.content_objects`

`gooddata_sdk.catalog.data_source.
entity_model.data_source`

`gooddata_sdk.catalog.data_source.entity_model.content_objects`

Modules

`gooddata_sdk.catalog.data_source.
entity_model.content_objects.table`

`gooddata_sdk.catalog.data_source.entity_model.content_objects.table`

Classes

`CatalogDataSourceTable`(**, id, type, attributes*)

`CatalogDataSourceTableAttributes`(**, columns*)

`CatalogDataSourceTableColumn`(**, name,
data_type*)

`gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTable`

```
class gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTable(*,
                                                                                               id:
                                                                                               str,
                                                                                               type:
                                                                                               str,
                                                                                               at-
                                                                                               tributes:
                                                                                               Cat-
                                                                                               a-
                                                                                               log-
                                                                                               Data-
                                                                                               Sourc-
                                                                                               eTableA-
                                                                                               tributes)
```

Bases: *Base*

__init__(**id*: str, *type*: str, *attributes*: CatalogDataSourceTableAttributes) → None

Method generated by attrs for class CatalogDataSourceTable.

Methods

<code>__init__(*, id, type, attributes)</code>	Method generated by attrs for class CatalogDataSourceTable.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>type</code>
<code>attributes</code>

classmethod from_api(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case*: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTableAttributes

```
class gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTableAttribu
```

Bases: *Base*

```
__init__(*, columns: List[CatalogDataSourceTableColumn], name_prefix: Optional[str] = None, path:
Optional[List[str]] = None, type: Optional[str] = None) → None
```

Method generated by attrs for class CatalogDataSourceTableAttributes.

Methods

<code><i>__init__</i>(*, columns[, name_prefix, path, type])</code>	Method generated by attrs for class CatalogDataSourceTableAttributes.
<code>client_class()</code>	
<code><i>from_api</i>(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code><i>from_dict</i>(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code><i>to_dict</i>([camel_case])</code>	Converts object into dictionary.

Attributes

<code>columns</code>
<code>name_prefix</code>
<code>path</code>
<code>type</code>

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTableColumn`

`class gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTableColumn`

Bases: `Base`

`__init__`(**, name: str, data_type: str, is_primary_key: Optional[bool] = None, referenced_table_column: Optional[str] = None, referenced_table_id: Optional[str] = None*) → None

Method generated by attrs for class `CatalogDataSourceTableColumn`.

Methods

<code>__init__(*, name, data_type[, ...])</code>	Method generated by attrs for class <code>CatalogData-SourceTableColumn</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>name</code>
<code>data_type</code>
<code>is_primary_key</code>
<code>referenced_table_column</code>
<code>referenced_table_id</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.data_source.entity_model.data_source

Classes

BigQueryAttributes(project_id[, port])

CatalogDataSource(id, name, schema, credentials)

CatalogDataSourceBigQuery(id, name, schema, ...)

CatalogDataSourcePostgres(id, name, schema, ...)

CatalogDataSourceRedshift(id, name, schema, ...)

CatalogDataSourceSnowflake(id, name, schema,
...)

CatalogDataSourceVertica(id, name, schema, ...)

DatabaseAttributes()

PostgresAttributes(host, db_name[, port])

RedshiftAttributes(host, db_name[, port])

SnowflakeAttributes(account, warehouse,
db_name)

VerticaAttributes(host, db_name[, port])

gooddata_sdk.catalog.data_source.entity_model.data_source.BigQueryAttributes

```
class gooddata_sdk.catalog.data_source.entity_model.data_source.BigQueryAttributes(project_id:  
                                                    str,  
                                                    port: str  
                                                    =  
                                                    '443')
```

Bases: *DatabaseAttributes*

```
__init__(project_id: str, port: str = '443')
```

Methods

```
__init__(project_id[, port])
```

Attributes

str_attributes

gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSource

```
class gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSource(id: str,
                                                                              name:
                                                                              str,
                                                                              schema:
                                                                              str, credentials:
                                                                              Credentials, url:
                                                                              Optional[str]
                                                                              = None,
                                                                              data_source_type:
                                                                              Optional[str]
                                                                              = None,
                                                                              db_specific_attributes:
                                                                              Optional[DatabaseAttributes]
                                                                              = None,
                                                                              enable_caching:
                                                                              Optional[bool]
                                                                              = None,
                                                                              cache_path:
                                                                              Optional[list[str]]
                                                                              = None,
                                                                              url_params:
                                                                              Optional[List[Tuple[str,
                                                                              str]]] =
                                                                              None)
```

Bases: *CatalogNameEntity*

```
__init__(id: str, name: str, schema: str, credentials: Credentials, url: Optional[str] = None,
          data_source_type: Optional[str] = None, db_specific_attributes: Optional[DatabaseAttributes] =
          None, enable_caching: Optional[bool] = None, cache_path: Optional[list[str]] = None,
          url_params: Optional[List[Tuple[str, str]]] = None)
```

Methods

`__init__(id, name, schema, credentials[, ...])`

`from_api(entity)`

`to_api()`

`to_api_patch(data_source_id, attributes)`

`gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceBigQuery`

```

class gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceBigQuery(id:
    str,
    name:
    str,
    schema:
    str,
    cre-
den-
tials:
    Credentials,
    url:
    Optional[str]
    =
    None,
    data_source_type:
    Optional[str]
    =
    None,
    db_specific_attrib
    Optional[DatabaseAttributes]
    =
    None,
    en-
able_caching:
    Optional[bool]
    =
    None,
    cache_path:
    Optional[list[str]]
    =
    None,
    url_params:
    Optional[List[Tuple[
    str]]]
    =
    None)

```

Bases: `CatalogDataSource`

```

__init__(id: str, name: str, schema: str, credentials: Credentials, url: Optional[str] = None,
    data_source_type: Optional[str] = None, db_specific_attributes: Optional[DatabaseAttributes] =
    None, enable_caching: Optional[bool] = None, cache_path: Optional[list[str]] = None,
    url_params: Optional[List[Tuple[str, str]]] = None)

```

Methods

`__init__(id, name, schema, credentials[, ...])`

`from_api(entity)`

`to_api()`

`to_api_patch(data_source_id, attributes)`

`gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourcePostgres`


```

class gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourcePostgres(id:
    str,
    name:
    str,
    schema:
    str,
    cre-
    den-
    tials:
    Credentials,
    url:
    Optional[str]
    =
    None,
    data_source_type:
    Optional[str]
    =
    None,
    db_specific_attrib
    utes:
    Optional[DatabaseAttributes]
    =
    None,
    enable_caching:
    Optional[bool]
    =
    None,
    cache_path:
    Optional[list[str]]
    =
    None,
    url_params:
    Optional[List[Tuple[
    str, str]]]
    =
    None)

```

Bases: *CatalogDataSource*

```

__init__(id: str, name: str, schema: str, credentials: Credentials, url: Optional[str] = None,
    data_source_type: Optional[str] = None, db_specific_attributes: Optional[DatabaseAttributes] =
    None, enable_caching: Optional[bool] = None, cache_path: Optional[list[str]] = None,
    url_params: Optional[List[Tuple[str, str]]] = None)

```

Methods

`__init__(id, name, schema, credentials[, ...])`

`from_api(entity)`

`to_api()`

`to_api_patch(data_source_id, attributes)`

`gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceRedshift`

```

class gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceRedshift(id:
    str,
    name:
    str,
    schema:
    str,
    cre-
den-
tials:
    Credentials,
    url:
    Optional[str]
    =
    None,
    data_source_type:
    Optional[str]
    =
    None,
    db_specific_attrib
    Optional[DatabaseAttributes]
    =
    None,
    en-
able_caching:
    Optional[bool]
    =
    None,
    cache_path:
    Optional[list[str]]
    =
    None,
    url_params:
    Optional[List[Tuple[str, str]]]
    =
    None)

```

Bases: [CatalogDataSourcePostgres](#)

```

__init__(id: str, name: str, schema: str, credentials: Credentials, url: Optional[str] = None,
    data_source_type: Optional[str] = None, db_specific_attributes: Optional[DatabaseAttributes] =
    None, enable_caching: Optional[bool] = None, cache_path: Optional[list[str]] = None,
    url_params: Optional[List[Tuple[str, str]]] = None)

```

Methods

`__init__(id, name, schema, credentials[, ...])`

`from_api(entity)`

`to_api()`

`to_api_patch(data_source_id, attributes)`

`gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceSnowflake`

```

class gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceSnowflake(id:
    str,
    name:
    str,
    schema:
    str,
    cre-
den-
tials:
    Credentials,
    url:
    Optional[str]
    =
    None,
    data_source_type:
    Optional[str]
    =
    None,
    db_specific_attr:
    Optional[DatabaseAttributes]
    =
    None,
    enable_caching:
    Optional[bool]
    =
    None,
    cache_path:
    Optional[list[str]]
    =
    None,
    url_params:
    Optional[List[Tuple[str, str]]]
    =
    None)

```

Bases: `CatalogDataSource`

```

__init__(id: str, name: str, schema: str, credentials: Credentials, url: Optional[str] = None,
    data_source_type: Optional[str] = None, db_specific_attributes: Optional[DatabaseAttributes] =
    None, enable_caching: Optional[bool] = None, cache_path: Optional[list[str]] = None,
    url_params: Optional[List[Tuple[str, str]]] = None)

```

Methods

`__init__(id, name, schema, credentials[, ...])`

`from_api(entity)`

`to_api()`

`to_api_patch(data_source_id, attributes)`

`gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceVertica`

```

class gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceVertica(id:
    str,
    name:
    str,
    schema:
    str,
    cre-
    den-
    tials:
    Cre-
    den-
    tials,
    url:
    Op-
    tional[str]
    =
    None,
    data_source_type:
    Op-
    tional[str]
    =
    None,
    db_specific_attribu-
    tional[DatabaseAttri-
    butes]
    =
    None,
    enable_caching:
    Op-
    tional[bool]
    =
    None,
    cache_path:
    Op-
    tional[list[str]]
    =
    None,
    url_params:
    Op-
    tional[List[Tuple[s-
    tr, str]]]
    =
    None)

```

Bases: [CatalogDataSourcePostgres](#)

```

__init__(id: str, name: str, schema: str, credentials: Credentials, url: Optional[str] = None,
    data_source_type: Optional[str] = None, db_specific_attributes: Optional[DatabaseAttributes] =
    None, enable_caching: Optional[bool] = None, cache_path: Optional[list[str]] = None,
    url_params: Optional[List[Tuple[str, str]]] = None)

```

Methods

```
__init__(id, name, schema, credentials[, ...])
```

```
from_api(entity)
```

```
to_api()
```

```
to_api_patch(data_source_id, attributes)
```

`gooddata_sdk.catalog.data_source.entity_model.data_source.DatabaseAttributes`

```
class gooddata_sdk.catalog.data_source.entity_model.data_source.DatabaseAttributes
```

```
Bases: object
```

```
__init__()
```

Methods

```
__init__()
```

Attributes

```
str_attributes
```

`gooddata_sdk.catalog.data_source.entity_model.data_source.PostgresAttributes`

```
class gooddata_sdk.catalog.data_source.entity_model.data_source.PostgresAttributes(host: str, db_name: str, port: str = '5432')
```

```
Bases: DatabaseAttributes
```

```
__init__(host: str, db_name: str, port: str = '5432')
```


Methods

```
__init__(host, db_name[, port])
```

Attributes

```
str_attributes
```

gooddata_sdk.catalog.data_source.entity_model.data_source.RedshiftAttributes

```
class gooddata_sdk.catalog.data_source.entity_model.data_source.RedshiftAttributes(host:
                                                                                   str,
                                                                                   db_name:
                                                                                   str,
                                                                                   port: str
                                                                                   =
                                                                                   '5439')
```

Bases: *PostgresAttributes*

```
__init__(host: str, db_name: str, port: str = '5439')
```

Methods

```
__init__(host, db_name[, port])
```

Attributes

```
str_attributes
```

gooddata_sdk.catalog.data_source.entity_model.data_source.SnowflakeAttributes

```
class gooddata_sdk.catalog.data_source.entity_model.data_source.SnowflakeAttributes(account:
                                                                                   str,
                                                                                   ware-
                                                                                   house:
                                                                                   str,
                                                                                   db_name:
                                                                                   str,
                                                                                   port:
                                                                                   str =
                                                                                   '443')
```

Bases: *DatabaseAttributes*

`__init__(account: str, warehouse: str, db_name: str, port: str = '443')`

Methods

`__init__(account, warehouse, db_name[, port])`

Attributes

`str_attributes`

`gooddata_sdk.catalog.data_source.entity_model.data_source.VerticaAttributes`

`class gooddata_sdk.catalog.data_source.entity_model.data_source.VerticaAttributes`*(host: str, db_name: str, port: str = '5433')*

Bases: *PostgresAttributes*

`__init__(host: str, db_name: str, port: str = '5433')`

Methods

`__init__(host, db_name[, port])`

Attributes

`str_attributes`

`gooddata_sdk.catalog.data_source.service`

Classes

`CatalogDataSourceService`(api_client)

gooddata_sdk.catalog.data_source.service.CatalogDataSourceService

```
class gooddata_sdk.catalog.data_source.service.CatalogDataSourceService(api_client:  
                                                                    GoodDataApiClient)
```

Bases: *CatalogServiceBase*

__init__(*api_client*: GoodDataApiClient) → None

Methods

`__init__(api_client)`

`create_or_update_data_source(data_source)`

`data_source_folder(data_source_id, ...)`

`delete_data_source(data_source_id)`

`generate_logical_model(data_source_id[, ...])`

`get_data_source(data_source_id)`

`get_declarative_data_sources()`

`get_declarative_pdm(data_source_id)`

`get_organization()`

`layout_organization_folder(layout_root_path)`

`list_data_source_tables(data_source_id)`

`list_data_sources()`

`load_and_put_declarative_data_sources(...)`

`load_and_put_declarative_pdm(data_source_id)`

`load_declarative_data_sources([layout_root_path])`

`load_declarative_pdm(data_source_id[, ...])`

`patch_data_source_attributes(data_source_id,
...)`

`put_declarative_data_sources(...[, ...])`

`put_declarative_pdm(data_source_id, ...)`

`register_upload_notification(data_source_id)`

`report_warnings(warnings)`

`scan_and_put_pdm(data_source_id[,
scan_request])`

`scan_data_source(data_source_id[, ...])`

`scan_schemata(data_source_id)`

`store_declarative_data_sources(...)`

`store_declarative_pdm(data_source_id[, ...])`

`test_data_sources_connection(...[, ...])`

Attributes

organization_id

gooddata_sdk.catalog.data_source.validation**Modules**

gooddata_sdk.catalog.data_source.validation.data_source

gooddata_sdk.catalog.data_source.validation.data_source**Classes**

DataSourceValidator(data_source_service)

gooddata_sdk.catalog.data_source.validation.data_source.DataSourceValidator

class gooddata_sdk.catalog.data_source.validation.data_source.DataSourceValidator(*data_source_service*: Catalog-Data-Source-Service)

Bases: object

__init__(*data_source_service*: CatalogDataSourceService)

Methods

__init__(data_source_service)

validate_data_source_ids(data_source_ids)

validate_ldm(model)

gooddata_sdk.catalog.entity

Classes

BasicCredentials(username, password)

CatalogEntity(entity)

CatalogNameEntity(id, name)

CatalogTitleEntity(id, title)

CatalogTypeEntity(id, type)

Credentials()

TokenCredentials(token)

TokenCredentialsFromFile(file_path)

gooddata_sdk.catalog.entity.BasicCredentials

class gooddata_sdk.catalog.entity.**BasicCredentials**(username: str, password: str)

Bases: *Credentials*

__init__(username: str, password: str)

Methods

__init__(username, password)

create(creds_classes, entity)

from_api(attributes)

is_part_of_api(entity)

to_api_args()

validate_instance(creds_classes, instance)

Attributes

PASSWORD_KEY

USER_KEY

gooddata_sdk.catalog.entity.CatalogEntity**class** gooddata_sdk.catalog.entity.CatalogEntity(*entity: dict[str, Any]*)

Bases: object

__init__(*entity: dict[str, Any]*) → None**Methods**

__init__(*entity*)

Attributes

description

id

obj_id

title

type

gooddata_sdk.catalog.entity.CatalogNameEntity**class** gooddata_sdk.catalog.entity.CatalogNameEntity(*id: str, name: str*)

Bases: object

__init__(*id: str, name: str*)

Methods

`__init__(id, name)`

`gooddata_sdk.catalog.entity.CatalogTitleEntity`

`class gooddata_sdk.catalog.entity.CatalogTitleEntity(id: str, title: str)`

Bases: object

`__init__(id: str, title: str)`

Methods

`__init__(id, title)`

`from_api(entity)`

`gooddata_sdk.catalog.entity.CatalogTypeEntity`

`class gooddata_sdk.catalog.entity.CatalogTypeEntity(id: str, type: str)`

Bases: object

`__init__(id: str, type: str)`

Methods

`__init__(id, type)`

`from_api(entity)`

`gooddata_sdk.catalog.entity.Credentials`

`class gooddata_sdk.catalog.entity.Credentials`

Bases: object

`__init__()`

Methods

`__init__()`

`create(creds_classes, entity)`

`from_api(entity)`

`is_part_of_api(entity)`

`to_api_args()`

`validate_instance(creds_classes, instance)`

`gooddata_sdk.catalog.entity.TokenCredentials`

class `gooddata_sdk.catalog.entity.TokenCredentials`(*token: str*)

Bases: `Credentials`

`__init__`(*token: str*)

Methods

`__init__(token)`

`create(creds_classes, entity)`

`from_api(entity)`

`is_part_of_api(entity)`

`to_api_args()`

`validate_instance(creds_classes, instance)`

Attributes

`TOKEN_KEY`

`USER_KEY`

gooddata_sdk.catalog.entity.TokenCredentialsFromFile

class gooddata_sdk.catalog.entity.TokenCredentialsFromFile(*file_path: Path*)

Bases: *Credentials*

__init__(*file_path: Path*)

Methods

__init__(*file_path*)

create(*creds_classes, entity*)

from_api(*entity*)

is_part_of_api(*entity*)

to_api_args()

token_from_file(*file_path*)

validate_instance(*creds_classes, instance*)

Attributes

TOKEN_KEY

USER_KEY

gooddata_sdk.catalog.identifier

Classes

CatalogAssigneeIdentifier(**, id, type*)

CatalogGrainIdentifier(**, id, type*)

CatalogLabelIdentifier(**, id, type*)

CatalogReferenceIdentifier(**, id*)

CatalogUserGroupIdentifier(**, id, type*)

CatalogWorkspaceIdentifier(**, id*)

gooddata_sdk.catalog.identifier.CatalogAssigneeIdentifier

class gooddata_sdk.catalog.identifier.CatalogAssigneeIdentifier(*, id: str, type: str)

Bases: *Base*

__init__(*, id: str, type: str) → None

Method generated by attrs for class CatalogAssigneeIdentifier.

Methods

<code>__init__(*, id, type)</code>	Method generated by attrs for class CatalogAssigneeIdentifier.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>type</code>

classmethod `from_api(entity: Dict[str, Any])` → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True)` → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.identifier.CatalogGrainIdentifier

class gooddata_sdk.catalog.identifier.CatalogGrainIdentifier(*, id: str, type: str)

Bases: *Base*

__init__(*, id: str, type: str) → None

Method generated by attrs for class CatalogGrainIdentifier.

Methods

<code>__init__(*, id, type)</code>	Method generated by attrs for class Catalog-GrainIdentifier.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>	
<code>type</code>	

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.identifier.CatalogLabelIdentifier`

class `gooddata_sdk.catalog.identifier.CatalogLabelIdentifier(*, id: str, type: str)`

Bases: `Base`

`__init__(*, id: str, type: str) → None`

Method generated by attrs for class CatalogLabelIdentifier.

Methods

<code>__init__(*, id, type)</code>	Method generated by attrs for class <code>CatalogLabelIdentifier</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>	
<code>type</code>	

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.identifier.CatalogReferenceIdentifier`

class `gooddata_sdk.catalog.identifier.CatalogReferenceIdentifier(*, id: str)`

Bases: `Base`

`__init__(*, id: str) → None`

Method generated by attrs for class `CatalogReferenceIdentifier`.

Methods

<code>__init__(*, id)</code>	Method generated by attrs for class <code>CatalogReferenceIdentifier</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>	
-----------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.identifier.CatalogUserGroupIdentifier`

class `gooddata_sdk.catalog.identifier.CatalogUserGroupIdentifier(*, id: str, type: str)`

Bases: `Base`

`__init__(*, id: str, type: str) → None`

Method generated by attrs for class `CatalogUserGroupIdentifier`.

Methods

<code>__init__(*, id, type)</code>	Method generated by attrs for class <code>CatalogUserGroupIdentifier</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

id

type

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.identifier.CatalogWorkspaceIdentifier`

class `gooddata_sdk.catalog.identifier.CatalogWorkspaceIdentifier`(**, id: str*)

Bases: `Base`

__init__(**, id: str*) → None

Method generated by attrs for class `CatalogWorkspaceIdentifier`.

Methods

<code>__init__</code> (* <i>, id</i>)	Method generated by attrs for class <code>CatalogWorkspaceIdentifier</code> .
--	---

`client_class`()

<code>from_api</code> (<i>entity</i>)	Creates object from entity passed by client class, which represents it as dictionary.
---	---

<code>from_dict</code> (<i>data[, camel_case]</i>)	Creates object from dictionary.
--	---------------------------------

`to_api`()

<code>to_dict</code> (<i>[camel_case]</i>)	Converts object into dictionary.
--	----------------------------------

Attributes

id

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

`to_dict`(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.organization

Modules

`gooddata_sdk.catalog.organization.`

`entity_model`

`gooddata_sdk.catalog.organization.service`

gooddata_sdk.catalog.organization.entity_model

Modules

`gooddata_sdk.catalog.organization.`

`entity_model.organization`

gooddata_sdk.catalog.organization.entity_model.organization

Classes

`CatalogOrganization`(* id, attributes)

`CatalogOrganizationAttributes`(*[, name, ...])

`CatalogOrganizationDocument`(* data)

gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganization

```
class gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganization(*,
                                                                                       id:
                                                                                       str,
                                                                                       at-
                                                                                       tributes:
                                                                                       Cat-
                                                                                       alo-
                                                                                       gOr-
                                                                                       ga-
                                                                                       ni-
                                                                                       za-
                                                                                       tion-
                                                                                       At-
                                                                                       tributes)
```


Bases: *Base*

__init__(**id*: str, *attributes*: CatalogOrganizationAttributes) → None

Method generated by attrs for class CatalogOrganization.

Methods

<code>__init__(*, id, attributes)</code>	Method generated by attrs for class CatalogOrganization.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>	
<code>attributes</code>	

classmethod from_api(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case*: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganizationAttributes

```

class gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganizationAttributes(*,
                                                                                               name:
                                                                                               Op-
                                                                                               tional[str]
                                                                                               =
                                                                                               None,
                                                                                               host-
                                                                                               name:
                                                                                               Op-
                                                                                               tional[str]
                                                                                               =
                                                                                               None,
                                                                                               al-
                                                                                               lowed_or-
                                                                                               iginal_iss-
                                                                                               uer_location:
                                                                                               Op-
                                                                                               tional[List[str]]
                                                                                               =
                                                                                               None,
                                                                                               oauth_iss-
                                                                                               uer_location:
                                                                                               Op-
                                                                                               tional[str]
                                                                                               =
                                                                                               None,
                                                                                               oauth_cli-
                                                                                               ent_id:
                                                                                               Op-
                                                                                               tional[str]
                                                                                               =
                                                                                               None)

```

Bases: *Base*

```

__init__(*, name: Optional[str] = None, hostname: Optional[str] = None, allowed_origins:
Optional[List[str]] = None, oauth_issuer_location: Optional[str] = None, oauth_client_id:
Optional[str] = None) → None

```

Method generated by attrs for class CatalogOrganizationAttributes.

Methods

<code>__init__(*[, name, hostname, ...])</code>	Method generated by attrs for class CatalogOrganizationAttributes.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

name

hostname

allowed_origins

oauth_issuer_location

oauth_client_id

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any]*, *camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganizationDocument`

```
class gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganizationDocument(*,
                                                                                               data:
                                                                                               Cat-
                                                                                               a-
                                                                                               l-
                                                                                               o-
                                                                                               gOr-
                                                                                               ga-
                                                                                               ni-
                                                                                               za-
                                                                                               tion)
```

Bases: `Base`

__init__(**, data: CatalogOrganization*) → None

Method generated by attrs for class `CatalogOrganizationDocument`.

Methods

<code>__init__(*, data)</code>	Method generated by attrs for class <code>CatalogOrganizationDocument</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api([oauth_client_secret])</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>data</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.organization.service`

Classes

<code>CatalogOrganizationService(api_client)</code>

`gooddata_sdk.catalog.organization.service.CatalogOrganizationService`

class `gooddata_sdk.catalog.organization.service.CatalogOrganizationService(api_client: GoodDataApiClient)`

Bases: `CatalogServiceBase`

__init__(api_client: GoodDataApiClient) → None

Methods

`__init__(api_client)`

`get_organization()`

`layout_organization_folder(layout_root_path)`

`update_name(name)`

`update_oidc_parameters(...)`

Attributes

`organization_id`

gooddata_sdk.catalog.permission**Modules**

`gooddata_sdk.catalog.permission.``declarative_model`

`gooddata_sdk.catalog.permission.service`

gooddata_sdk.catalog.permission.declarative_model**Modules**

`gooddata_sdk.catalog.permission.``declarative_model.permission`

gooddata_sdk.catalog.permission.declarative_model.permission**Classes**

`CatalogDeclarativeDataSourcePermission(*,
...)`

`CatalogDeclarativeSingleWorkspacePermission(*,
...)`

`CatalogDeclarativeWorkspaceHierarchyPermission(*,
...)`

`CatalogDeclarativeWorkspacePermissions(*[,
...])`

gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeDataSourcePermission

```
class gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeDataSourcePermission
```

Bases: *Base*

__init__(**name*: str, *assignee*: CatalogAssigneeIdentifier) → None

Method generated by attrs for class CatalogDeclarativeDataSourcePermission.

Methods

<code>__init__(*, name, assignee)</code>	Method generated by attrs for class CatalogDeclarativeDataSourcePermission.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>name</code>	
<code>assignee</code>	

classmethod from_api(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case*: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeSingleWorkspacePermission**class** gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeSingleWorkspacePerBases: *Base***__init__**(**name*: str, *assignee*: CatalogAssigneeIdentifier) → None

Method generated by attrs for class CatalogDeclarativeSingleWorkspacePermission.

Methods

<i>__init__</i> (* <i>name</i> , <i>assignee</i>)	Method generated by attrs for class CatalogDeclarativeSingleWorkspacePermission.
<i>client_class</i> ()	
<i>from_api</i> (<i>entity</i>)	Creates object from entity passed by client class, which represents it as dictionary.
<i>from_dict</i> (<i>data</i> [, <i>camel_case</i>])	Creates object from dictionary.
<i>to_api</i> ()	
<i>to_dict</i> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

<i>name</i>
<i>assignee</i>

classmethod **from_api**(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod **from_dict**(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case*: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeWorkspaceHierarchyPermission`

`class gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeWorkspaceHierarchyPermission`

Bases: *Base*

`__init__`(**name*: str, *assignee*: CatalogAssigneeIdentifier) → None

Method generated by attrs for class CatalogDeclarativeWorkspaceHierarchyPermission.

Methods

<code>__init__</code> (* <i>name</i> , <i>assignee</i>)	Method generated by attrs for class CatalogDeclarativeWorkspaceHierarchyPermission.
<code>client_class</code> ()	
<code>from_api</code> (<i>entity</i>)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (<i>data</i> [, <i>camel_case</i>])	Creates object from dictionary.
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

<code>name</code>
<code>assignee</code>

classmethod `from_api`(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case*: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeWorkspacePermissions

```
class gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativeWorkspacePermissions
```

Bases: *Base*

```
__init__(*[, permissions: List[CatalogDeclarativeSingleWorkspacePermission] = NOTHING,
          hierarchy_permissions: List[CatalogDeclarativeWorkspaceHierarchyPermission] = NOTHING)
    → None
```

Method generated by attrs for class CatalogDeclarativeWorkspacePermissions.

Methods

<code>__init__(*[, permissions, hierarchy_permissions])</code>	Method generated by attrs for class CatalogDeclarativeWorkspacePermissions.
<code>client_class()</code>	
<code><i>from_api</i>(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code><i>from_dict</i>(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code><i>to_dict</i>([camel_case])</code>	Converts object into dictionary.

Attributes

<code>permissions</code>	
<code>hierarchy_permissions</code>	

classmethod `from_api`(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

`to_dict`(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.permission.service

Classes

CatalogPermissionService(api_client)

gooddata_sdk.catalog.permission.service.CatalogPermissionService

class gooddata_sdk.catalog.permission.service.CatalogPermissionService(*api_client:*
GoodDataApiClient)

Bases: *CatalogServiceBase*

`__init__`(*api_client: GoodDataApiClient*) → None

Methods

`__init__`(api_client)

`get_declarative_permissions`(workspace_id)

`get_organization`()

`layout_organization_folder`(layout_root_path)

`put_declarative_permissions`(workspace_id,
...)

Attributes

`organization_id`

gooddata_sdk.catalog.setting**Classes**

CatalogDeclarativeSetting(* , id[, content])

gooddata_sdk.catalog.setting.CatalogDeclarativeSetting

class gooddata_sdk.catalog.setting.CatalogDeclarativeSetting(*, id: str, content: Optional[Dict[str, Any]] = None)

Bases: *Base*

__init__(* , id: str, content: Optional[Dict[str, Any]] = None) → None

Method generated by attrs for class CatalogDeclarativeSetting.

Methods

<i>__init__</i> (* , id[, content])	Method generated by attrs for class CatalogDeclarativeSetting.
<i>client_class</i> ()	
<i>from_api</i> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<i>from_dict</i> (data[, camel_case])	Creates object from dictionary.
<i>to_api</i> ()	
<i>to_dict</i> ([camel_case])	Converts object into dictionary.

Attributes

<i>id</i>
<i>content</i>

classmethod *from_api*(entity: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod *from_dict*(data: Dict[str, Any], camel_case: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.types

gooddata_sdk.catalog.user

Modules

gooddata_sdk.catalog.user.declarative_model

gooddata_sdk.catalog.user.entity_model

gooddata_sdk.catalog.user.service

gooddata_sdk.catalog.user.declarative_model

Modules

gooddata_sdk.catalog.user.declarative_model.user

gooddata_sdk.catalog.user.declarative_model.user_and_user_groups

gooddata_sdk.catalog.user.declarative_model.user_group

gooddata_sdk.catalog.user.declarative_model.user

Classes

CatalogDeclarativeUser(, id[, auth_id, ...])*

CatalogDeclarativeUsers(, users)*

gooddata_sdk.catalog.user.declarative_model.user.CatalogDeclarativeUser

```
class gooddata_sdk.catalog.user.declarative_model.user.CatalogDeclarativeUser(*, id: str,
                                                                              auth_id:
                                                                              Optional[str]
                                                                              = None,
                                                                              user_groups:
                                                                              List[CatalogUserGroupIdentifier]
                                                                              = NOTHING,
                                                                              settings:
                                                                              List[CatalogDeclarativeSetting]
                                                                              = NOTHING)
```

Bases: *Base*

```
__init__(* , id: str, auth_id: Optional[str] = None, user_groups: List[CatalogUserGroupIdentifier] = NOTHING, settings: List[CatalogDeclarativeSetting] = NOTHING) → None
```

Method generated by attrs for class CatalogDeclarativeUser.

Methods

<code>__init__(* , id[, auth_id, user_groups, settings])</code>	Method generated by attrs for class CatalogDeclarativeUser.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>auth_id</code>
<code>user_groups</code>
<code>settings</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.user.declarative_model.user.CatalogDeclarativeUsers`

```
class gooddata_sdk.catalog.user.declarative_model.user.CatalogDeclarativeUsers(* , users: List[CatalogDeclarativeUser])
```

Bases: `Base`

```
__init__(* , users: List[CatalogDeclarativeUser]) → None
```

Method generated by attrs for class CatalogDeclarativeUsers.

Methods

<code>__init__(*, users)</code>	Method generated by attrs for class <code>CatalogDeclarativeUsers</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(layout_organization_folder)</code>	
<code>store_to_disk(layout_organization_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>users</code>	
--------------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.user.declarative_model.user_and_user_groups`

Classes

<code>CatalogDeclarativeUsersUserGroups(*, users, ...)</code>

`gooddata_sdk.catalog.user.declarative_model.user_and_user_groups.CatalogDeclarativeUsersUserGroups`

class `gooddata_sdk.catalog.user.declarative_model.user_and_user_groups.CatalogDeclarativeUsersUserGroups`

Bases: `Base`

`__init__`(*, *users*: List[CatalogDeclarativeUser], *user_groups*: List[CatalogDeclarativeUserGroup]) → None

Method generated by attrs for class CatalogDeclarativeUsersUserGroups.

Methods

<code>__init__</code> (*, <i>users</i> , <i>user_groups</i>)	Method generated by attrs for class CatalogDeclarativeUsersUserGroups.
<code>client_class</code> ()	
<code>from_api</code> (<i>entity</i>)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (<i>data</i> [, <i>camel_case</i>])	Creates object from dictionary.
<code>load_from_disk</code> (<i>layout_organization_folder</i>)	
<code>store_to_disk</code> (<i>layout_organization_folder</i>)	
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

<code>users</code>	
<code>user_groups</code>	

classmethod `from_api`(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case*: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.declarative_model.user_group

Classes

<code>CatalogDeclarativeUserGroup</code> (*, <i>id</i> [, <i>parents</i>])
<code>CatalogDeclarativeUserGroups</code> (*[, <i>user_groups</i>])

gooddata_sdk.catalog.user.declarative_model.user_group.CatalogDeclarativeUserGroup

```
class gooddata_sdk.catalog.user.declarative_model.user_group.CatalogDeclarativeUserGroup(*,
                                                                                          id:
                                                                                          str,
                                                                                          par-
                                                                                          ents:
                                                                                          Op-
                                                                                          tional[List[Catalog
                                                                                          =
                                                                                          None])
```

Bases: *Base*

__init__(*, id: str, parents: Optional[List[CatalogUserGroupIdentifier]] = None) → None

Method generated by attrs for class CatalogDeclarativeUserGroup.

Methods

<code>__init__(*, id[, parents])</code>	Method generated by attrs for class CatalogDeclarativeUserGroup.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>parents</code>

classmethod from_api(entity: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(data: Dict[str, Any], camel_case: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.declarative_model.user_group.CatalogDeclarativeUserGroups

```
class gooddata_sdk.catalog.user.declarative_model.user_group.CatalogDeclarativeUserGroups(*,
                                                                                          user_groups:
                                                                                          List[CatalogDeclarativeUserGroup] =
                                                                                          NOTHING)
    = NOTHING
```

Bases: *Base*

__init__(*, user_groups: List[CatalogDeclarativeUserGroup] = NOTHING) → None

Method generated by attrs for class CatalogDeclarativeUserGroups.

Methods

<code>__init__</code> (*[, user_groups])	Method generated by attrs for class CatalogDeclarativeUserGroups.
<code>client_class</code> ()	
<code>from_api</code> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (data[, camel_case])	Creates object from dictionary.
<code>load_from_disk</code> (layout_organization_folder)	
<code>store_to_disk</code> (layout_organization_folder)	
<code>to_api</code> ()	
<code>to_dict</code> ([camel_case])	Converts object into dictionary.

Attributes

<code>user_groups</code>

classmethod from_api(entity: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(data: Dict[str, Any], camel_case: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.entity_model

Modules

`gooddata_sdk.catalog.user.entity_model.
user`

`gooddata_sdk.catalog.user.entity_model.
user_group`

gooddata_sdk.catalog.user.entity_model.user

Classes

`CatalogUser(*, id[, attributes, relationships])`

`CatalogUserAttributes(*[, authentication_id])`

`CatalogUserDocument(*, data)`

`CatalogUserGroupsData(*[, data])`

`CatalogUserRelationships(*[, user_groups])`

gooddata_sdk.catalog.user.entity_model.user.CatalogUser

```
class gooddata_sdk.catalog.user.entity_model.user.CatalogUser(*, id: str, attributes:
    Optional[CatalogUserAttributes] =
    None, relationships: Op-
    tional[CatalogUserRelationships]
    = None)
```

Bases: `Base`

```
__init__(*, id: str, attributes: Optional[CatalogUserAttributes] = None, relationships:
    Optional[CatalogUserRelationships] = None) → None
```

Method generated by attrs for class `CatalogUser`.

Methods

<code>__init__(*, id[, attributes, relationships])</code> <code>client_class()</code>	Method generated by attrs for class CatalogUser.
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code> <code>init(user_id[, authentication_id, ...])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>get_user_groups</code>
<code>id</code>
<code>attributes</code>
<code>relationships</code>

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any]*, *camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.user.entity_model.user.CatalogUserAttributes`

class `gooddata_sdk.catalog.user.entity_model.user.CatalogUserAttributes`(*, *authentication_id: Optional[str] = None*)

Bases: `Base`

__init__(*, *authentication_id: Optional[str] = None*) → None

Method generated by attrs for class CatalogUserAttributes.

Methods

<code>__init__(*[, authentication_id])</code>	Method generated by attrs for class <code>CatalogUserAttributes</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>authentication_id</code>	
--------------------------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.user.entity_model.user.CatalogUserDocument`

class `gooddata_sdk.catalog.user.entity_model.user.CatalogUserDocument(*, data: CatalogUser)`

Bases: `Base`

`__init__(*, data: CatalogUser) → None`

Method generated by attrs for class `CatalogUserDocument`.

Methods

<code>__init__(*, data)</code>	Method generated by attrs for class CatalogUserDocument.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>init(user_id[, authentication_id, ...])</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.
<code>update_user([authentication_id, user_group_ids])</code>	

Attributes

<code>data</code>	
-------------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.user.entity_model.user.CatalogUserGroupsData`

class `gooddata_sdk.catalog.user.entity_model.user.CatalogUserGroupsData(*, data: Optional[List[CatalogUserGroup]] = None)`

Bases: `Base`

__init__(*, data: Optional[List[CatalogUserGroup]] = None) → None

Method generated by attrs for class CatalogUserGroupsData.

Methods

<code>__init__(*[, data])</code>	Method generated by attrs for class CatalogUserGroupsData.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>get_user_groups</code>
<code>data</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.user.entity_model.user.CatalogUserRelationships`

```
class gooddata_sdk.catalog.user.entity_model.user.CatalogUserRelationships(*, user_groups:
    Optional[CatalogUserGroupsData]
    = None)
```

Bases: `Base`

__init__(*, user_groups: Optional[CatalogUserGroupsData] = None) → None

Method generated by attrs for class CatalogUserRelationships.

Methods

<code>__init__(*[, user_groups])</code>	Method generated by attrs for class <code>CatalogUserRelationships</code> .
<code>client_class()</code>	
<code>create_user_relationships(user_group_ids)</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>get_user_groups</code>	
<code>user_groups</code>	

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.entity_model.user_group

Classes

<code>CatalogUserGroup(*, id[, relationships])</code>
<code>CatalogUserGroupDocument(*, data)</code>
<code>CatalogUserGroupParents(*[, data])</code>
<code>CatalogUserGroupRelationships(*[, parents])</code>

gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroup

```
class gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroup(*, id: str,
                                                                    relationships: Optional[CatalogUserGroupRelationships],
                                                                    = None)
```

Bases: *Base*

```
__init__(*, id: str, relationships: Optional[CatalogUserGroupRelationships] = None) → None
```

Method generated by attrs for class CatalogUserGroup.

Methods

<code>__init__(*, id[, relationships])</code>	Method generated by attrs for class CatalogUserGroup.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>init(user_group_id[, user_group_parent_ids])</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>get_parents</code>	
<code>id</code>	
<code>relationships</code>	

```
classmethod from_api(entity: Dict[str, Any]) → T
```

Creates object from entity passed by client class, which represents it as dictionary.

```
classmethod from_dict(data: Dict[str, Any], camel_case: bool = True) → T
```

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

```
to_dict(camel_case: bool = True) → Dict[str, Any]
```

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroupDocument

```
class gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroupDocument(*, data: CatalogUserGroup)
```

Bases: *Base*

```
__init__(*, data: CatalogUserGroup) → None
```

Method generated by attrs for class CatalogUserGroupDocument.

Methods

<code>__init__(*, data)</code>	Method generated by attrs for class CatalogUserGroupDocument.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>init(user_group_id[, user_group_parent_ids])</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.
<code>update_user_group([user_group_parents_id])</code>	

Attributes

<code>data</code>

```
classmethod from_api(entity: Dict[str, Any]) → T
```

Creates object from entity passed by client class, which represents it as dictionary.

```
classmethod from_dict(data: Dict[str, Any], camel_case: bool = True) → T
```

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

```
to_dict(camel_case: bool = True) → Dict[str, Any]
```

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroupParents

```
class gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroupParents(*, data: Optional[List[CatalogUserGroupParents]] = None)
```

Bases: *Base*

```
__init__(*, data: Optional[List[CatalogUserGroupParents]] = None) → None
```

Method generated by attrs for class CatalogUserGroupParents.

Methods

<code>__init__(*[, data])</code>	Method generated by attrs for class CatalogUserGroupParents.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>get_parents</code>
<code>data</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroupRelationships

```
class gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroupRelationships(*, parents: Optional[CatalogUserGroupRelationships]) = None
```

Bases: *Base*

__init__(*[, parents: *Optional[CatalogUserGroupParents]* = None) → None

Method generated by attrs for class CatalogUserGroupRelationships.

Methods

<code>__init__(*[, parents])</code>	Method generated by attrs for class CatalogUserGroupRelationships.
<code>client_class()</code>	
<code>create_user_group_relationships(...)</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>get_parents</code>
<code>parents</code>

classmethod from_api(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(*data: Dict[str, Any]*, *camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.user.service

Classes

<code>CatalogUserService(api_client)</code>

gooddata_sdk.catalog.user.service.CatalogUserService

class gooddata_sdk.catalog.user.service.CatalogUserService(*api_client*: GoodDataApiClient)

Bases: *CatalogServiceBase*

__init__(*api_client*: GoodDataApiClient) → None

Methods

`__init__(api_client)`

`create_or_update_user(user)`

`create_or_update_user_group(user_group)`

`delete_user(user_id)`

`delete_user_group(user_group_id)`

`get_declarative_user_groups()`

`get_declarative_users()`

`get_declarative_users_user_groups()`

`get_organization()`

`get_user(user_id)`

`get_user_group(user_group_id)`

`layout_organization_folder(layout_root_path)`

`list_user_groups()`

`list_users()`

`load_and_put_declarative_user_groups([...])`

`load_and_put_declarative_users([...])`

`load_and_put_declarative_users_user_groups([...])`

`load_declarative_user_groups([layout_root_path])`

`load_declarative_users([layout_root_path])`

`load_declarative_users_user_groups([...])`

`put_declarative_user_groups(user_groups)`

`put_declarative_users(users)`

`put_declarative_users_user_groups(...)`

`store_declarative_user_groups([layout_root_path])`

`store_declarative_users([layout_root_path])`

`store_declarative_users_user_groups([...])`

Attributes

organization_id

gooddata_sdk.catalog.workspace

Modules

*gooddata_sdk.catalog.workspace.
content_service*

*gooddata_sdk.catalog.workspace.
declarative_model*

*gooddata_sdk.catalog.workspace.
entity_model*

*gooddata_sdk.catalog.workspace.
model_container*

gooddata_sdk.catalog.workspace.service

gooddata_sdk.catalog.workspace.content_service

Classes

CatalogWorkspaceContentService(api_client)

gooddata_sdk.catalog.workspace.content_service.CatalogWorkspaceContentService

class gooddata_sdk.catalog.workspace.content_service.CatalogWorkspaceContentService(*api_client:*
Good-
DataApi-
Client)

Bases: *CatalogServiceBase*

__init__(*api_client:* GoodDataApiClient) → None

Methods

<code>__init__(api_client)</code>	
<code>compute_valid_objects(workspace_id, ctx)</code>	Returns attributes, facts, and metrics which are valid to add to a context that already contains some entities from the semantic model.
<code>get_attributes_catalog(workspace_id)</code>	
<code>get_declarative_analytics_model(workspace_id)</code>	
<code>get_declarative_ldm(workspace_id)</code>	
<code>get_dependent_entities_graph(workspace_id)</code>	
<code>get_dependent_entities_graph_from_entry_points(...)</code>	
<code>get_facts_catalog(workspace_id)</code>	
<code>get_full_catalog(workspace_id)</code>	Retrieves catalog for a workspace.
<code>get_labels_catalog(workspace_id)</code>	
<code>get_metrics_catalog(workspace_id)</code>	
<code>get_organization()</code>	
<code>layout_organization_folder(layout_root_path)</code>	
<code>layout_workspace_folder(workspace_id, ...)</code>	
<code>load_and_put_declarative_analytics_model(...)</code>	
<code>load_and_put_declarative_ldm(workspace_id[, ...])</code>	
<code>load_declarative_analytics_model(workspace_id)</code>	
<code>load_declarative_ldm(workspace_id[, ...])</code>	
<code>put_declarative_analytics_model(...)</code>	
<code>put_declarative_ldm(workspace_id, ldm[, ...])</code>	
<code>store_declarative_analytics_model(workspace_id)</code>	
<code>store_declarative_ldm(workspace_id[, ...])</code>	

Attributes

organization_id

compute_valid_objects(*workspace_id*: str, *ctx*: Union[Attribute, Metric, Filter, CatalogLabel, CatalogFact, CatalogMetric, List[Union[Attribute, Metric, Filter, CatalogLabel, CatalogFact, CatalogMetric]], ExecutionDefinition]) → Dict[str, Set[str]]

Returns attributes, facts, and metrics which are valid to add to a context that already contains some entities from the semantic model. The entities are typically used to compute analytics and come from the execution definition. You may, however, specify the entities through different layers of convenience.

Parameters

- **workspace_id** – workspace identifier
- **ctx** – items already in context. you can specify context in one of the following ways:
 - single item or list of items from the execution model
 - single item or list of items from catalog model; catalog fact, label or metric may be added
 - the entire execution definition that is used to compute analytics

Returns

a dict of sets; type of available object is used as key in the dict, the value is a set containing id's of available items

get_full_catalog(*workspace_id*: str) → *CatalogWorkspaceContent*

Retrieves catalog for a workspace. Catalog contains all data sets and metrics defined in that workspace.

Parameters

workspace_id – workspace identifier

gooddata_sdk.catalog.workspace.declarative_model

Modules

gooddata_sdk.catalog.workspace.declarative_model.workspace

gooddata_sdk.catalog.workspace.declarative_model.workspace

Modules

gooddata_sdk.catalog.workspace.declarative_model.analytics_model

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model

gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace

`gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model`

Modules

`gooddata_sdk.catalog.workspace.
declarative_model.workspace.
analytics_model.analytics_model`

`gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model`

Classes

`CatalogAnalyticsBase(*, id)`

`CatalogDeclarativeAnalyticalDashboard(*, id,
...)`

`CatalogDeclarativeAnalytics(*[, analytics])`

`CatalogDeclarativeAnalyticsLayer(*[, ...])`

`CatalogDeclarativeDashboardPlugin(*, id, ...)`

`CatalogDeclarativeFilterContext(*, id, ...)`

`CatalogDeclarativeMetric(*, id, title, content)`

`CatalogDeclarativeVisualizationObject(*, id,
...)`

`gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogAnalyticsBase`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogAnalyticsBase`

Bases: `Base`

`__init__(*, id: str) → None`

Method generated by attrs for class `CatalogAnalyticsBase`.

Methods

<code>__init__(*, id)</code>	Method generated by attrs for class CatalogAnalyticsBase.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(analytics_file)</code>	
<code>store_to_disk(analytics_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>	
-----------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeAnalyticalDashboard`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeAnalyticalDashboard`

Bases: `CatalogAnalyticsBase`

`__init__`(**id*: str, *title*: str, *content*: Dict[str, Any], *description*: Optional[str] = None, *tags*: Optional[List[str]] = None) → None

Method generated by attrs for class `CatalogDeclarativeAnalyticalDashboard`.

Methods

<code>__init__</code> (* <i>id</i> , <i>title</i> , <i>content</i> [, ...])	Method generated by attrs for class <code>CatalogDeclarativeAnalyticalDashboard</code> .
<code>client_class</code> ()	
<code>from_api</code> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (<i>data</i> [, <i>camel_case</i>])	Creates object from dictionary.
<code>load_from_disk</code> (<i>analytics_file</i>)	
<code>store_to_disk</code> (<i>analytics_folder</i>)	
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

id

title

content

description

tags

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeAnalyticsLayer

class `gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeAnalyticsLayer`

Bases: *Base*

__init__(**, analytics: Optional[CatalogDeclarativeAnalyticsLayer] = None*) → None

Method generated by attrs for class CatalogDeclarativeAnalyticsLayer.

Methods

<code>__init__(*[, analytics])</code>	Method generated by attrs for class CatalogDeclarativeAnalytics.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(workspace_folder)</code>	
<code>store_to_disk(workspace_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>analytics</code>	
------------------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeAnalyticsLayer`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeAnalyticsLayer`

Bases: *Base*

```
__init__(*, analytical_dashboards: List[CatalogDeclarativeAnalyticalDashboard] = NOTHING, dashboard_plugins: List[CatalogDeclarativeDashboardPlugin] = NOTHING, filter_contexts: List[CatalogDeclarativeFilterContext] = NOTHING, metrics: List[CatalogDeclarativeMetric] = NOTHING, visualization_objects: List[CatalogDeclarativeVisualizationObject] = NOTHING) → None
```

Method generated by attrs for class `CatalogDeclarativeAnalyticsLayer`.

Methods

<code>__init__(*[, analytical_dashboards, ...])</code>	Method generated by attrs for class CatalogDeclarativeAnalyticsLayer.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>get_analytical_dashboards_folder(...)</code>	
<code>get_analytics_model_folder(workspace_folder)</code>	
<code>get_dashboard_plugins_folder(...)</code>	
<code>get_filter_contexts_folder(...)</code>	
<code>get_metrics_folder(analytics_model_folder)</code>	
<code>get_visualization_objects_folder(...)</code>	
<code>load_from_disk(workspace_folder)</code>	
<code>store_to_disk(workspace_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>analytical_dashboards</code>
<code>dashboard_plugins</code>
<code>filter_contexts</code>
<code>metrics</code>
<code>visualization_objects</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeDashboardPlugin`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeDashboardPlugin`

Bases: `CatalogAnalyticsBase`

`__init__`(*, *id*: str, *title*: str, *content*: Dict[str, Any], *description*: Optional[str] = None, *tags*: Optional[List[str]] = None) → None

Method generated by attrs for class `CatalogDeclarativeDashboardPlugin`.

Methods

<code>__init__</code> (*, <i>id</i> , <i>title</i> , <i>content</i> [, ...])	Method generated by attrs for class <code>CatalogDeclarativeDashboardPlugin</code> .
<code>client_class</code> ()	
<code>from_api</code> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (data[, <i>camel_case</i>])	Creates object from dictionary.
<code>load_from_disk</code> (analytics_file)	
<code>store_to_disk</code> (analytics_folder)	
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

`id`

`title`

`content`

`description`

`tags`

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeFilterContext**class** gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeFilterContextBases: *CatalogAnalyticsBase***__init__**(**, id: str, title: str, content: Dict[str, Any], description: Optional[str] = None, tags: Optional[List[str]] = None*) → None

Method generated by attrs for class CatalogDeclarativeFilterContext.

Methods

<code>__init__(*, id, title, content[, ...])</code>	Method generated by attrs for class <code>CatalogDeclarativeFilterContext</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(analytics_file)</code>	
<code>store_to_disk(analytics_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>title</code>
<code>content</code>
<code>description</code>
<code>tags</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeFilterContext`

```
class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.Catalog
```

Bases: *CatalogAnalyticsBase*

```
__init__(*, id: str, title: str, content: Dict[str, Any], description: Optional[str] = None, tags:
Optional[List[str]] = None) → None
```

Method generated by attrs for class CatalogDeclarativeMetric.

Methods

<code><i>__init__</i>(*, id, title, content[, ...])</code>	Method generated by attrs for class CatalogDeclarativeMetric.
<code>client_class()</code>	
<code><i>from_api</i>(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code><i>from_dict</i>(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(analytics_file)</code>	
<code>store_to_disk(analytics_folder)</code>	
<code>to_api()</code>	
<code><i>to_dict</i>([camel_case])</code>	Converts object into dictionary.

Attributes

id

title

content

description

tags

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeVisualizationObject

class `gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeVisualizationObject`

Bases: *CatalogAnalyticsBase*

__init__(**, id: str, title: str, content: Dict[str, Any], description: Optional[str] = None, tags: Optional[List[str]] = None*) → None

Method generated by attrs for class `CatalogDeclarativeVisualizationObject`.

Methods

<code>__init__(*, id, title, content[, ...])</code>	Method generated by attrs for class <code>CatalogDeclarativeVisualizationObject</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(analytics_file)</code>	
<code>store_to_disk(analytics_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>title</code>
<code>content</code>
<code>description</code>
<code>tags</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model

Modules

*gooddata_sdk.catalog.workspace.
declarative_model.workspace.logical_model.
dataset*

*gooddata_sdk.catalog.workspace.
declarative_model.workspace.logical_model.
date_dataset*

*gooddata_sdk.catalog.workspace.
declarative_model.workspace.logical_model.
ldm*

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset

Modules

*gooddata_sdk.catalog.workspace.
declarative_model.workspace.logical_model.
dataset.dataset*

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset

Classes

CatalogDataSourceTableIdentifier(, id, ...)*

CatalogDeclarativeAttribute(, id, title, ...)*

CatalogDeclarativeDataset(, id, title, ...)*

CatalogDeclarativeFact(, id, title, ..., ...)*

CatalogDeclarativeLabel(, id, title, ..., ...)*

CatalogDeclarativeReference(, identifier, ...)*

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDataSourceTableIdentifier

class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDataSourceTableIdentifier

Bases: *Base*

__init__(*, id: str, data_source_id: str) → None

Method generated by attrs for class CatalogDataSourceTableIdentifier.

Methods

<code>__init__(*, id, data_source_id)</code>	Method generated by attrs for class <code>CatalogDataSourceTableIdentifier</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>data_source_id</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarative`

```
class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogD
```

Bases: *Base*

```
__init__(* , id: str, title: str, source_column: str, labels: List[CatalogDeclarativeLabel], default_view:
Optional[CatalogLabelIdentifier] = None, sort_column: Optional[str] = None, sort_direction:
Optional[str] = None, description: Optional[str] = None, tags: Optional[List[str]] = None) →
None
```

Method generated by attrs for class CatalogDeclarativeAttribute.

Methods

<code>__init__(*, id, title, source_column, labels)</code>	Method generated by attrs for class CatalogDeclarativeAttribute.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>title</code>
<code>source_column</code>
<code>labels</code>
<code>default_view</code>
<code>sort_column</code>
<code>sort_direction</code>
<code>description</code>
<code>tags</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(`camel_case: bool = True`) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarative`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogD`

Bases: *Base*

```
__init__(*, id: str, title: str, grain: List[CatalogGrainIdentifier], references: List[CatalogDeclarativeReference], description: Optional[str] = None, attributes: Optional[List[CatalogDeclarativeAttribute]] = None, facts: Optional[List[CatalogDeclarativeFact]] = None, data_source_table_id: Optional[CatalogDataSourceTableIdentifier] = None, tags: Optional[List[str]] = None) → None
```

Method generated by attrs for class `CatalogDeclarativeDataset`.

Methods

<code>__init__(*, id, title, grain, references[, ...])</code>	Method generated by attrs for class CatalogDeclarativeDataset.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(dataset_file)</code>	
<code>store_to_disk(datasets_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>title</code>
<code>grain</code>
<code>references</code>
<code>description</code>
<code>attributes</code>
<code>facts</code>
<code>data_source_table_id</code>
<code>tags</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarative`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogD`

Bases: *Base*

`__init__`(*, *id*: str, *title*: str, *source_column*: str, *description*: Optional[str] = None, *tags*: Optional[List[str]] = None) → None

Method generated by attrs for class CatalogDeclarativeFact.

Methods

<code>__init__</code> (*, <i>id</i> , <i>title</i> , <i>source_column</i> [, ...])	Method generated by attrs for class CatalogDeclarativeFact.
<code>client_class</code> ()	
<code>from_api</code> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (data[, <i>camel_case</i>])	Creates object from dictionary.
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

id

title

source_column

description

tags

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarative

class `gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogD`

Bases: [Base](#)

`__init__`(**id*: str, *title*: str, *source_column*: str, *description*: Optional[str] = None, *tags*: Optional[List[str]] = None, *value_type*: Optional[str] = None) → None

Method generated by attrs for class CatalogDeclarativeLabel.

Methods

<code>__init__</code> (* <i>id</i> , <i>title</i> , <i>source_column</i> [, ...])	Method generated by attrs for class CatalogDeclarativeLabel.
<code>client_class</code> ()	
<code>from_api</code> (<i>entity</i>)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (<i>data</i> [, <i>camel_case</i>])	Creates object from dictionary.
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

<code>id</code>
<code>title</code>
<code>source_column</code>
<code>description</code>
<code>tags</code>
<code>value_type</code>

classmethod `from_api`(*entity*: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data*: Dict[str, Any], *camel_case*: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case*: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarative`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogD`

Bases: *Base*

`__init__`(*, *identifier*: *CatalogReferenceIdentifier*, *multivalued*: *bool*, *source_columns*: *List[str]*) → None
 Method generated by attrs for class *CatalogDeclarativeReference*.

Methods

<code>__init__</code> (*, <i>identifier</i> , <i>multivalued</i> , ...)	Method generated by attrs for class <i>CatalogDeclarativeReference</i> .
<code>client_class</code> ()	
<code>from_api</code> (<i>entity</i>)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (<i>data</i> [, <i>camel_case</i>])	Creates object from dictionary.
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

<code>identifier</code>
<code>multivalued</code>
<code>source_columns</code>

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset

Modules

*gooddata_sdk.catalog.workspace.
declarative_model.workspace.logical_model.
date_dataset.date_dataset*

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset

Classes

CatalogDeclarativeDateDataset(, id, title, ...)*

CatalogGranularitiesFormatting(, ...)*

gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset.Catalog


```
class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset
```

Bases: *Base*

```
__init__(* , id: str, title: str, granularities_formatting: CatalogGranularitiesFormatting, granularities:  
List[str], description: Optional[str] = None, tags: Optional[List[str]] = None) → None
```

Method generated by attrs for class CatalogDeclarativeDateDataset.

Methods

<code>__init__(*, id, title, ..., description, tags)</code>	Method generated by attrs for class CatalogDeclarativeDateDataset.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(date_instance_file)</code>	
<code>store_to_disk(date_instances_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>title</code>
<code>granularities_formatting</code>
<code>granularities</code>
<code>description</code>
<code>tags</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict `(camel_case: bool = True) → Dict[str, Any]`

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset.Catalog`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset`

Bases: *Base*

`__init__(*, title_base: str, title_pattern: str) → None`

Method generated by attrs for class `CatalogGranularitiesFormatting`.

Methods

<code>__init__(*, title_base, title_pattern)</code>	Method generated by attrs for class <code>CatalogGranularitiesFormatting</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>title_base</code>
<code>title_pattern</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm`

Classes

`CatalogDeclarativeLdm(*[, datasets, ...])`

`CatalogDeclarativeModel(*[, ldm])`

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm.CatalogDeclarativeLdm`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm.CatalogDeclarativeLdm`

Bases: `Base`

`__init__(*, datasets: List[CatalogDeclarativeDataset] = NOTHING, date_instances: List[CatalogDeclarativeDateDataset] = NOTHING) → None`

Method generated by attrs for class `CatalogDeclarativeLdm`.

Methods

<code>__init__(*[, datasets, date_instances])</code>	Method generated by attrs for class <code>CatalogDeclarativeLdm</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>get_datasets_folder(ldm_folder)</code>	
<code>get_date_instances_folder(ldm_folder)</code>	
<code>get_ldm_folder(workspace_folder)</code>	
<code>load_from_disk(workspace_folder)</code>	
<code>store_to_disk(workspace_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

datasets

date_instances

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any]*, *camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm.CatalogDeclarativeModel`

class `gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm.CatalogDeclarativeModel`

Bases: *Base*

__init__(**, ldm: Optional[CatalogDeclarativeLdm] = None*) → None

Method generated by attrs for class CatalogDeclarativeModel.

Methods

`__init__`(**, ldm*) Method generated by attrs for class CatalogDeclarativeModel.

`client_class`()

`from_api`(*entity*) Creates object from entity passed by client class, which represents it as dictionary.

`from_dict`(*data*[, *camel_case*]) Creates object from dictionary.

`load_from_disk`(*workspace_folder*)

`modify_mapped_data_source`(*data_source_mapping*)

`store_to_disk`(*workspace_folder*)

`to_api`()

`to_dict`([*camel_case*]) Converts object into dictionary.

Attributes

ldm

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace

Functions

`get_workspace_folder`(workspace_id, ...)

gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.get_workspace_folder

`gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.get_workspace_folder`(*workspace_id: str, lay-out_organization: Path*) → Path

Classes

`CatalogDeclarativeWorkspace`(* , id, name[, ...])

`CatalogDeclarativeWorkspaceDataFilter`(* , id, ...)

`CatalogDeclarativeWorkspaceDataFilterSetting`(* , ...)

`CatalogDeclarativeWorkspaceDataFilters`(* , ...)

`CatalogDeclarativeWorkspaceModel`(*[, ldm, ...])

`CatalogDeclarativeWorkspaces`(* , workspaces, ...)

`gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspace`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspace(`

Bases: `Base`

```
__init__(*, id: str, name: str, model: Optional[CatalogDeclarativeWorkspaceModel] = None, parent:
Optional[CatalogWorkspaceIdentifier] = None, permissions:
List[CatalogDeclarativeSingleWorkspacePermission] = NOTHING, hierarchy_permissions:
List[CatalogDeclarativeWorkspaceHierarchyPermission] = NOTHING, early_access:
Optional[str] = None, settings: List[CatalogDeclarativeSetting] = NOTHING) → None
```

Method generated by attrs for class `CatalogDeclarativeWorkspace`.

Methods

<code>__init__(*, id, name[, model, parent, ...])</code>	Method generated by attrs for class CatalogDeclarativeWorkspace.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(workspaces_folder, workspace_id)</code>	
<code>store_to_disk(workspaces_folder)</code>	
<code>to_api([include_nested_structures])</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>name</code>
<code>model</code>
<code>parent</code>
<code>permissions</code>
<code>hierarchy_permissions</code>
<code>early_access</code>
<code>settings</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict `(camel_case: bool = True) → Dict[str, Any]`

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilter`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilter`

Bases: *Base*

`__init__`(**id*: *str*, *title*: *str*, *column_name*: *str*, *workspace_data_filter_settings*: *List*[*CatalogDeclarativeWorkspaceDataFilterSetting*], *description*: *Optional*[*str*] = *None*, *workspace*: *Optional*[*CatalogWorkspaceIdentifier*] = *None*) → *None*

Method generated by attrs for class `CatalogDeclarativeWorkspaceDataFilter`.

Methods

<code>__init__</code> (* <i>id</i> , <i>title</i> , <i>column_name</i> , ...[, ...])	Method generated by attrs for class <code>CatalogDeclarativeWorkspaceDataFilter</code> .
<code>client_class</code> ()	
<code>from_api</code> (<i>entity</i>)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (<i>data</i> [, <i>camel_case</i>])	
	param data Data loaded for example from the file.
<code>load_from_disk</code> (<i>workspaces_data_filter_file</i>)	
<code>store_to_disk</code> (<i>workspaces_data_filters_folder</i>)	
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

id

title

column_name

workspace_data_filter_settings

description

workspace

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: dict[str, Any], camel_case: bool = True*) → *CatalogDeclarativeWorkspaceDataFilter*

Parameters

- **data** – Data loaded for example from the file.
- **camel_case** – True if the variable names in the input data are serialized names as specified in the OpenAPI document. False if the variables names in the input data are python variable names in PEP-8 snake case.

Returns

CatalogDeclarativeWorkspaceDataFilter object.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilterSetting`

`class gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilterSetting`

Bases: `Base`

`__init__`(*, *id*: str, *title*: str, *filter_values*: List[str], *workspace*: CatalogWorkspaceIdentifier, *description*: Optional[str] = None) → None

Method generated by attrs for class `CatalogDeclarativeWorkspaceDataFilterSetting`.

Methods

<code>__init__</code> (*, <i>id</i> , <i>title</i> , <i>filter_values</i> , <i>workspace</i>)	Method generated by attrs for class <code>CatalogDeclarativeWorkspaceDataFilterSetting</code> .
<code>client_class</code> ()	
<code>from_api</code> (entity)	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict</code> (data[, <i>camel_case</i>])	Creates object from dictionary.
<code>to_api</code> ()	
<code>to_dict</code> ([<i>camel_case</i>])	Converts object into dictionary.

Attributes

id
title
filter_values
workspace
description

classmethod from_api(entity: Dict[str, Any]) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod from_dict(data: Dict[str, Any], camel_case: bool = True) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilter

class gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilter

Bases: *Base*

__init__(* , workspace_data_filters: List[CatalogDeclarativeWorkspaceDataFilter]) → None

Method generated by attrs for class CatalogDeclarativeWorkspaceDataFilters.

Methods

__init__ (* , workspace_data_filters)	Method generated by attrs for class CatalogDeclarativeWorkspaceDataFilters.
client_class ()	
from_api (entity)	Creates object from entity passed by client class, which represents it as dictionary.
from_dict (data[, camel_case])	Creates object from dictionary.
load_from_disk (layout_organization_folder)	
store_to_disk (layout_organization_folder)	
to_api ()	
to_dict ([camel_case])	Converts object into dictionary.

Attributes

`workspace_data_filters`

classmethod `from_api`(*entity: Dict[str, Any]*) → T

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict`(*data: Dict[str, Any], camel_case: bool = True*) → T

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(*camel_case: bool = True*) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceModel

class `gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceModel`

Bases: `Base`

__init__(**, ldm: Optional[CatalogDeclarativeLdm] = None, analytics: Optional[CatalogDeclarativeAnalyticsLayer] = None*) → None

Method generated by attrs for class `CatalogDeclarativeWorkspaceModel`.

Methods

<code>__init__(*[, ldm, analytics])</code>	Method generated by attrs for class <code>CatalogDeclarativeWorkspaceModel</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(workspace_folder)</code>	
<code>store_to_disk(workspace_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>ldm</code>	
<code>analytics</code>	

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaces`

class `gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaces`

Bases: `Base`

__init__(*[, workspaces: List[CatalogDeclarativeWorkspace], workspace_data_filters: List[CatalogDeclarativeWorkspaceDataFilter]) → None

Method generated by attrs for class `CatalogDeclarativeWorkspaces`.

Methods

<code>__init__(*, workspaces, workspace_data_filters)</code>	Method generated by attrs for class CatalogDeclarativeWorkspaces.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>load_from_disk(layout_organization_folder)</code>	
<code>store_to_disk(layout_organization_folder)</code>	
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.
<code>workspace_data_filters_folder(...)</code>	
<code>workspaces_folder(layout_organization_folder)</code>	

Attributes

<code>workspaces</code>
<code>workspace_data_filters</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.entity_model`

Modules

<code>gooddata_sdk.catalog.workspace.entity_model.content_objects</code>
<code>gooddata_sdk.catalog.workspace.entity_model.graph_objects</code>
<code>gooddata_sdk.catalog.workspace.entity_model.workspace</code>

gooddata_sdk.catalog.workspace.entity_model.content_objects

Modules

*gooddata_sdk.catalog.workspace.
entity_model.content_objects.dataset*

*gooddata_sdk.catalog.workspace.
entity_model.content_objects.metric*

gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset

Classes

CatalogAttribute(entity, labels)

CatalogDataset(entity, attributes, facts)

CatalogFact(entity)

CatalogLabel(entity)

gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogAttribute

class gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogAttribute(*entity: dict[str, Any], labels: list[CatalogLabel]*)

Bases: *CatalogEntity*

__init__(*entity: dict[str, Any], labels: list[CatalogLabel]*) → None

Methods

__init__(entity, labels)

as_computable()

find_label(id_obj)

primary_label()

Attributes

dataset
description
granularity
id
labels
obj_id
title
type

gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogDataset

```
class gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogDataset(entity:
dict[str, Any],
at-
tributes:
list[CatalogAttribute],
facts:
list[CatalogFact])
```

Bases: *CatalogEntity*

__init__(*entity: dict[str, Any], attributes: list[CatalogAttribute], facts: list[CatalogFact]*) → None

Methods

<i>__init__</i> (entity, attributes, facts)	
<i>filter_dataset</i> (valid_objects)	Filters dataset so that it contains only attributes and facts that are part of the provided valid objects structure.
<i>find_label_attribute</i> (id_obj)	

Attributes

attributes

data_type

description

facts

id

obj_id

title

type

filter_dataset(*valid_objects: Dict[str, Set[str]]*) → Optional[*CatalogDataset*]

Filters dataset so that it contains only attributes and facts that are part of the provided valid objects structure.

Parameters

valid_objects – mapping of object type to a set of valid object ids

Returns

CatalogDataset containing only valid attributes and facts; None if all of the attributes and facts were filtered out

`gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogFact`

class `gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogFact`(*entity: dict[str, Any]*)

Bases: *CatalogEntity*

`__init__`(*entity: dict[str, Any]*) → None

Methods

`__init__`(*entity*)

`as_computable`()

Attributes

description

id

obj_id

title

type

gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogLabel

```
class gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset.CatalogLabel(entity:
                                                                    dict[str,
                                                                    Any])
```

Bases: *CatalogEntity*

`__init__`(entity: dict[str, Any]) → None

Methods

`__init__`(entity)

`as_computable`()

Attributes

description

id

obj_id

primary

title

type

`gooddata_sdk.catalog.workspace.entity_model.content_objects.metric`

Classes

`CatalogMetric(entity)`

`gooddata_sdk.catalog.workspace.entity_model.content_objects.metric.CatalogMetric`

`class gooddata_sdk.catalog.workspace.entity_model.content_objects.metric.CatalogMetric(entity: dict[str, Any])`

Bases: `CatalogEntity`

`__init__(entity: dict[str, Any]) → None`

Methods

`__init__(entity)`

`as_computable()`

Attributes

`description`

`format`

`id`

`obj_id`

`title`

`type`

gooddata_sdk.catalog.workspace.entity_model.graph_objects**Modules**

```
gooddata_sdk.catalog.workspace.  
entity_model.graph_objects.graph
```

gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph**Classes**

```
CatalogDependentEntitiesGraph(*[, nodes,  
edges])
```

```
CatalogDependentEntitiesNode(*, id, type[, ...])
```

```
CatalogDependentEntitiesRequest(*[, identi-  
fiers])
```

```
CatalogDependentEntitiesResponse(*, graph)
```

```
CatalogEntityIdentifier(*, id, type)
```

gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesGraph

```
class gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesGraph(*,  
node  
List[  
=  
NOTHING  
ING  
edge  
List[  
=  
NOTHING  
ING
```

Bases: *Base*

```
__init__(*, nodes: List[CatalogDependentEntitiesNode] = NOTHING, edges:  
List[List[CatalogEntityIdentifier]] = NOTHING) → None
```

Method generated by attrs for class CatalogDependentEntitiesGraph.

Methods

<code>__init__(*[, nodes, edges])</code>	Method generated by attrs for class <code>CatalogDependentEntitiesGraph</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>nodes</code>
<code>edges</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesNode`

```
class gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesNode(*,
                                                    id: str,
                                                    type: str,
                                                    title: Optional[str] = None)
```

Bases: `Base`

__init__(*, id: str, type: str, title: Optional[str] = None) → None

Method generated by attrs for class `CatalogDependentEntitiesNode`.

Methods

<code>__init__(*, id, type[, title])</code>	Method generated by attrs for class <code>CatalogDependentEntitiesNode</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>type</code>
<code>title</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesRequest`

class `gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesRequest(*,`

Bases: `Base`

__init__(*, `identifiers: List[CatalogEntityIdentifier] = NOTHING`) → None

Method generated by attrs for class `CatalogDependentEntitiesRequest`.

Methods

<code>__init__(*[, identifiers])</code>	Method generated by attrs for class <code>CatalogDependentEntitiesRequest</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>identifiers</code>	
--------------------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesResponse`

class `gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesResponse(`

Bases: `Base`

`__init__(*, graph: CatalogDependentEntitiesGraph) → None`

Method generated by attrs for class `CatalogDependentEntitiesResponse`.

Methods

<code>__init__(*, graph)</code>	Method generated by attrs for class CatalogDependentEntitiesResponse.
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>graph</code>	
--------------------	--

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogEntityIdentifier`

```
class gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogEntityIdentifier(*,
                                                                                               id:
                                                                                               str,
                                                                                               type:
                                                                                               str)
```

Bases: `Base`

__init__(*, id: str, type: str) → None

Method generated by attrs for class CatalogEntityIdentifier.

Methods

<code>__init__(*, id, type)</code>	Method generated by attrs for class <code>CatalogEntityIdentifier</code> .
<code>client_class()</code>	
<code>from_api(entity)</code>	Creates object from entity passed by client class, which represents it as dictionary.
<code>from_dict(data[, camel_case])</code>	Creates object from dictionary.
<code>to_api()</code>	
<code>to_dict([camel_case])</code>	Converts object into dictionary.

Attributes

<code>id</code>
<code>type</code>

classmethod `from_api(entity: Dict[str, Any]) → T`

Creates object from entity passed by client class, which represents it as dictionary.

classmethod `from_dict(data: Dict[str, Any], camel_case: bool = True) → T`

Creates object from dictionary. It needs to be specified if the dictionary is in camelCase or snake_case.

to_dict(camel_case: bool = True) → Dict[str, Any]

Converts object into dictionary. Optional argument if the dictionary should be camelCase or snake_case can be specified.

`gooddata_sdk.catalog.workspace.entity_model.workspace`

Classes

`CatalogWorkspace(workspace_id, name[, parent_id])`

`gooddata_sdk.catalog.workspace.entity_model.workspace.CatalogWorkspace`

class `gooddata_sdk.catalog.workspace.entity_model.workspace.CatalogWorkspace(workspace_id: str, name: str, parent_id: Optional[str] = None)`

Bases: `CatalogNameEntity`

`__init__(workspace_id: str, name: str, parent_id: Optional[str] = None)`

Methods

`__init__(workspace_id, name[, parent_id])`

`from_api(entity)`

`to_api()`

gooddata_sdk.catalog.workspace.model_container**Classes**

`CatalogWorkspaceContent(valid_obj_fun, ...)`

gooddata_sdk.catalog.workspace.model_container.CatalogWorkspaceContent

```
class gooddata_sdk.catalog.workspace.model_container.CatalogWorkspaceContent(
    valid_obj_fun:
        func-
        tools.partial[dict[str,
        set[str]]],
    datasets:
        list[CatalogDataset],
    metrics:
        list[CatalogMetric])
```

Bases: object

```
__init__(valid_obj_fun: functools.partial[dict[str, set[str]]], datasets: list[CatalogDataset], metrics:
        list[CatalogMetric]) → None
```

Methods

`__init__(valid_obj_fun, datasets, metrics)`

<code>catalog_with_valid_objects(ctx)</code>	Returns a new instance of catalog which contains only those datasets (attributes and facts) that are valid in the provided context.
--	---

`create_workspace_content_catalog(...)`

<code>find_label_attribute(id_obj)</code>	Get attribute by label id.
---	----------------------------

<code>get_dataset(dataset_id)</code>	Gets dataset by id.
--------------------------------------	---------------------

<code>get_metric(metric_id)</code>	Gets metric by id.
------------------------------------	--------------------

Attributes

datasets

metrics

catalog_with_valid_objects(*ctx*: Union[Attribute, Metric, Filter, CatalogLabel, CatalogFact, CatalogMetric, List[Union[Attribute, Metric, Filter, CatalogLabel, CatalogFact, CatalogMetric]], ExecutionDefinition]) → *CatalogWorkspaceContent*

Returns a new instance of catalog which contains only those datasets (attributes and facts) that are valid in the provided context. The context is composed of one more more entities of the semantic model and the filtered catalog will contain only those entities that can be safely added on top of that existing context.

Parameters

ctx – existing context. you can specify context in one of the following ways:

- single item or list of items from the execution model
- single item or list of items from catalog model; catalog fact, label or metric may be added
- the entire execution definition that is used to compute analytics

find_label_attribute(*id_obj*: Union[str, ObjId, Dict[str, Dict[str, str]], Dict[str, str]]) → Optional[*CatalogAttribute*]

Get attribute by label id.

get_dataset(*dataset_id*: Union[str, ObjId]) → Optional[*CatalogDataset*]

Gets dataset by id. The id can be either an instance of ObjId or string containing serialized ObjId ('dataset/some.dataset.id') or contain just the id part (some.dataset.id).

Parameters

dataset_id – fully qualified dataset entity id (type/id) or just the identifier of dataset entity

Returns

instance of CatalogDataset or None if no such dataset in catalog

Return type

CatalogDataset

get_metric(*metric_id*: Union[str, ObjId]) → Optional[*CatalogMetric*]

Gets metric by id. The id can be either an instance of ObjId or string containing serialized ObjId ('metric/some.metric.id') or contain just the id part ('some.metric.id').

Parameters

metric_id – fully qualified metric entity id (type/id) or just the identifier of metric entity

Returns

instance of CatalogMetric or None if no such metric in catalog

Return type

CatalogMetric

gooddata_sdk.catalog.workspace.service**Classes**

CatalogWorkspaceService(api_client)

gooddata_sdk.catalog.workspace.service.CatalogWorkspaceService**class** gooddata_sdk.catalog.workspace.service.**CatalogWorkspaceService**(*api_client*:
GoodDataApiClient)Bases: *CatalogServiceBase***__init__**(*api_client*: GoodDataApiClient) → None

Methods

<code>__init__(api_client)</code>	
<code>create_or_update(workspace)</code>	
<code>delete_workspace(workspace_id)</code>	This method is implemented according to our implementation of delete workspace, which returns HTTP 204 no matter if the workspace_id exists.
<code>get_declarative_workspace(workspace_id)</code>	
<code>get_declarative_workspace_data_filters()</code>	
<code>get_declarative_workspaces()</code>	
<code>get_organization()</code>	
<code>get_workspace(workspace_id)</code>	Gets workspace content and returns it as Catalog-Workspace object.
<code>layout_organization_folder(layout_root_path)</code>	
<code>list_workspaces()</code>	
<code>load_and_put_declarative_workspace(workspace_id)</code>	
<code>load_and_put_declarative_workspace_data_filters([...])</code>	
<code>load_and_put_declarative_workspaces([...])</code>	
<code>load_declarative_workspace(workspace_id[, ...])</code>	
<code>load_declarative_workspace_data_filters([...])</code>	
<code>load_declarative_workspaces([layout_root_path])</code>	
<code>put_declarative_workspace(workspace_id, ...)</code>	
<code>put_declarative_workspace_data_filters(...)</code>	
<code>put_declarative_workspaces(workspace)</code>	
<code>store_declarative_workspace(workspace_id[, ...])</code>	
<code>store_declarative_workspace_data_filters([...])</code>	
<code>store_declarative_workspaces([layout_root_path])</code>	

Attributes

organization_id

delete_workspace(workspace_id: str) → None

This method is implemented according to our implementation of delete workspace, which returns HTTP 204 no matter if the workspace_id exists.

get_workspace(workspace_id: str) → *CatalogWorkspace*

Gets workspace content and returns it as CatalogWorkspace object.

Parameters

workspace_id – An input string parameter of workspace id.

Returns

CatalogWorkspace object containing structure of workspace.

3.1.2 gooddata_sdk.client

Module containing a class that provides access to metadata and afm services.

Classes

<i>GoodDataApiClient</i> (host, token[, ...])	Provide access to metadata and afm services.
---	--

gooddata_sdk.client.GoodDataApiClient

class gooddata_sdk.client.**GoodDataApiClient**(host: str, token: str, custom_headers: Optional[dict[str, str]] = None, extra_user_agent: Optional[str] = None)

Bases: object

Provide access to metadata and afm services.

__init__(host: str, token: str, custom_headers: Optional[dict[str, str]] = None, extra_user_agent: Optional[str] = None) → None

Take url, token for connecting to GoodData.CN.

HTTP requests made by this class may be enriched by *custom_headers* dict containing header names as keys and header values as dict values.

extra_user_agent is optional string to be added to default http User-Agent header. This takes precedence over *custom_headers* setting.

Methods

<code>__init__(host, token[, custom_headers, ...])</code>	Take url, token for connecting to GoodData.CN.
---	--

Attributes

`afm_client`

`metadata_client`

`scan_client`

3.1.3 gooddata_sdk.compute

Modules

`gooddata_sdk.compute.model`

`gooddata_sdk.compute.service`

gooddata_sdk.compute.model

Modules

`gooddata_sdk.compute.model.attribute`

`gooddata_sdk.compute.model.base`

`gooddata_sdk.compute.model.execution`

`gooddata_sdk.compute.model.filter`

`gooddata_sdk.compute.model.metric`

gooddata_sdk.compute.model.attribute**Classes**

Attribute(local_id, label)

gooddata_sdk.compute.model.attribute.Attribute
class gooddata_sdk.compute.model.attribute.**Attribute**(local_id: str, label: Union[ObjId, str])

 Bases: *ExecModelEntity*
__init__(local_id: str, label: Union[ObjId, str]) → None

Creates new attribute that can be used to slice or dice metric values during computation.

Parameters

- **local_id** – identifier of the attribute within the execution
- **label** – identifier of the label to use for slicing or dicing; specified either as ObjId or str containing the label id

Methods

__init__(local_id, label) Creates new attribute that can be used to slice or dice metric values during computation.

as_api_model()

has_same_label(other)

Attributes

 label

 local_id

gooddata_sdk.compute.model.base**Classes**

ExecModelEntity()

Filter()

ObjId(id, type)

gooddata_sdk.compute.model.base.ExecModelEntity

class gooddata_sdk.compute.model.base.ExecModelEntity

Bases: object

`__init__()` → None

Methods

`__init__()`

`as_api_model()`

gooddata_sdk.compute.model.base.Filter

class gooddata_sdk.compute.model.base.Filter

Bases: *ExecModelEntity*

`__init__()` → None

Methods

`__init__()`

`as_api_model()`

`is_noop()`

Attributes

`apply_on_result`

gooddata_sdk.compute.model.base.ObjId

class gooddata_sdk.compute.model.base.ObjId(*id: str, type: str*)

Bases: object

`__init__(id: str, type: str)` → None

Methods

`__init__(id, type)`

`as_afm_id()`

`as_afm_id_attribute()`

`as_afm_id_dataset()`

`as_afm_id_label()`

`as_identifier()`

Attributes

`id`

`type`

gooddata_sdk.compute.model.execution

Functions

<code>compute_model_to_api_model([attributes, ...])</code>	Transforms categorized execution model entities (attributes, metrics, facts) into an API model that can be used for computations of data results or computations of object availability.
--	--

gooddata_sdk.compute.model.execution.compute_model_to_api_model

`gooddata_sdk.compute.model.execution.compute_model_to_api_model`(*attributes*:
Optional[*list*[*Attribute*]] =
None, *metrics*:
Optional[*list*[*Metric*]] = *None*,
filters: *Optional*[*list*[*Filter*]] =
None) → `models.AFM`

Transforms categorized execution model entities (attributes, metrics, facts) into an API model that can be used for computations of data results or computations of object availability.

Parameters

- **attributes** – optionally specify list of attributes
- **metrics** – optionally specify list of metrics
- **filters** – optionally specify list of filters

Classes

<i>BareExecutionResponse</i> (actions_api, ...)	Holds ExecutionResponse from triggered report computation and allows reading report's results.
<i>Execution</i> (actions_api, workspace_id, ...)	An envelope class holding execution related classes:
<i>ExecutionDefinition</i> (attributes, metrics, ...)	
<i>ExecutionResponse</i>	alias of <i>Execution</i>
<i>ExecutionResult</i> (result)	
<i>TotalDefinition</i> (local_id, aggregation, ...)	
<i>TotalDimension</i> (idx[, items])	

gooddata_sdk.compute.model.execution.BareExecutionResponse

```
class gooddata_sdk.compute.model.execution.BareExecutionResponse(actions_api: ActionsApi,
                                                                workspace_id: str, response:
                                                                AfmExecutionResponse)
```

Bases: object

Holds ExecutionResponse from triggered report computation and allows reading report's results.

```
__init__(actions_api: ActionsApi, workspace_id: str, response: AfmExecutionResponse)
```

Methods

<i>__init__</i> (actions_api, workspace_id, response)	
<i>read_result</i> (limit[, offset])	Reads from the execution result.

Attributes

dimensions
result_id
workspace_id

```
read_result(limit: Union[int, list[int]], offset: Union[None, int, list[int]] = None) → ExecutionResult
```

Reads from the execution result.

gooddata_sdk.compute.model.execution.Execution

```
class gooddata_sdk.compute.model.execution.Execution(actions_api: ActionsApi, workspace_id: str,
                                                    exec_def: ExecutionDefinition, response:
                                                    AfmExecutionResponse)
```

Bases: object

An envelope class holding execution related classes:

- exec_def ExecutionDefinition
- bare_exec_response BareExecutionResponse

```
__init__(actions_api: ActionsApi, workspace_id: str, exec_def: ExecutionDefinition, response:
          AfmExecutionResponse)
```

Methods

```
__init__(actions_api, workspace_id, ...)
```

```
read_result(limit[, offset])
```

Attributes

```
bare_exec_response
```

```
dimensions
```

```
exec_def
```

```
result_id
```

```
workspace_id
```

gooddata_sdk.compute.model.execution.ExecutionDefinition

```
class gooddata_sdk.compute.model.execution.ExecutionDefinition(attributes:
                                                                Optional[list[Attribute]], metrics:
                                                                Optional[list[Metric]], filters:
                                                                Optional[list[Filter]], dimensions:
                                                                list[Optional[list[str]]], totals:
                                                                Optional[list[TotalDefinition]] =
                                                                None)
```

Bases: object

```
__init__(attributes: Optional[list[Attribute]], metrics: Optional[list[Metric]], filters: Optional[list[Filter]],
          dimensions: list[Optional[list[str]]], totals: Optional[list[TotalDefinition]] = None) → None
```

Methods

`__init__(attributes, metrics, filters, ...)`

`as_api_model()`

`has_attributes()`

`has_filters()`

`has_metrics()`

`is_one_dim()`

`is_two_dim()`

Attributes

`attributes`

`dimensions`

`filters`

`metrics`

`gooddata_sdk.compute.model.execution.ExecutionResponse`

`gooddata_sdk.compute.model.execution.ExecutionResponse`

alias of *Execution*

`gooddata_sdk.compute.model.execution.ExecutionResult`

class `gooddata_sdk.compute.model.execution.ExecutionResult`(*result: ExecutionResult*)

Bases: `object`

`__init__(result: ExecutionResult)`

Methods

`__init__(result)`

`check_size_limits(result_size_limits)`

`get_all_header_values(dim, header_idx)`

`get_all_headers(dim)`

`is_complete([dim])`

`next_page_start([dim])`

Attributes

`data`

`grand_totals`

`headers`

`paging`

`paging_count`

`paging_offset`

`paging_total`

`gooddata_sdk.compute.model.execution.TotalDefinition`

```
class gooddata_sdk.compute.model.execution.TotalDefinition(local_id: str, aggregation: str,  
metric_local_id: str, total_dims:  
list[TotalDimension])
```

Bases: object

```
__init__(local_id: str, aggregation: str, metric_local_id: str, total_dims: list[TotalDimension])
```

 → None

Method generated by attrs for class TotalDefinition.

Methods

<code>__init__(local_id, aggregation, ...)</code>	Method generated by attrs for class TotalDefinition.
---	--

Attributes

<code>local_id</code>	total's local identifier
<code>aggregation</code>	aggregation function; case insensitive; one of SUM, MIN, MAX, MED, AVG
<code>metric_local_id</code>	local identifier of the measure to calculate total for
<code>total_dims</code>	

aggregation: str

aggregation function; case insensitive; one of SUM, MIN, MAX, MED, AVG

local_id: str

total's local identifier

metric_local_id: str

local identifier of the measure to calculate total for

`gooddata_sdk.compute.model.execution.TotalDimension`

class `gooddata_sdk.compute.model.execution.TotalDimension`(*idx: int, items: list[str] = NOTHING*)

Bases: object

`__init__(idx: int, items: list[str] = NOTHING) → None`

Method generated by attrs for class TotalDimension.

Methods

<code>__init__(idx[, items])</code>	Method generated by attrs for class TotalDimension.
-------------------------------------	---

Attributes

<code>idx</code>	index of dimension in which to calculate the total
<code>items</code>	items to use during total calculation

idx: int

index of dimension in which to calculate the total

items: list[str]

items to use during total calculation

Exceptions

ResultSizeLimitsExceeded(result_size_limits, ...)

gooddata_sdk.compute.model.execution.ResultSizeLimitsExceeded

exception gooddata_sdk.compute.model.execution.ResultSizeLimitsExceeded(*result_size_limits*:
Tuple[*Optional*[*int*],
...],
actual_result_size:
Tuple[*Optional*[*int*],
...],
first_violating_index:
int)

gooddata_sdk.compute.model.filter

Classes

AbsoluteDateFilter(dataset, from_date, to_date)

AllTimeFilter() Filter that is semantically equivalent to absent filter.

AttributeFilter(label[, values])

MetricValueFilter(metric, operator, values)

NegativeAttributeFilter(label[, values])

PositiveAttributeFilter(label[, values])

RankingFilter(metrics, operator, value, ...)

RelativeDateFilter(dataset, granularity, ...)

gooddata_sdk.compute.model.filter.AbsoluteDateFilter

class gooddata_sdk.compute.model.filter.AbsoluteDateFilter(*dataset*: *ObjId*, *from_date*: *str*, *to_date*:
str)

Bases: *Filter*

__init__(*dataset*: *ObjId*, *from_date*: *str*, *to_date*: *str*) → None

Methods

`__init__(dataset, from_date, to_date)`

`as_api_model()`

`is_noop()`

Attributes

`apply_on_result`

`dataset`

`from_date`

`to_date`

`gooddata_sdk.compute.model.filter.AllTimeFilter`

class `gooddata_sdk.compute.model.filter.AllTimeFilter`

Bases: *Filter*

Filter that is semantically equivalent to absent filter.

This filter exists because ‘All time filter’ retrieved from GoodData.CN is non-standard as it does not have *from* and *to* fields; this is also the reason why `as_api_model` method is not implemented - it would lead to invalid object.

The main feature of this filter is `noop`.

`__init__()` → None

Methods

`__init__()`

`as_api_model()`

`is_noop()`

Attributes

apply_on_result

gooddata_sdk.compute.model.filter.AttributeFilter

```
class gooddata_sdk.compute.model.filter.AttributeFilter(label: Union[ObjId, str, Attribute], values: list[str] = None)
```

Bases: *Filter*

```
__init__(label: Union[ObjId, str, Attribute], values: list[str] = None) → None
```

Methods

```
__init__(label[, values])
```

```
as_api_model()
```

```
is_noop()
```

Attributes

apply_on_result

label

values

gooddata_sdk.compute.model.filter.MetricValueFilter

```
class gooddata_sdk.compute.model.filter.MetricValueFilter(metric: Union[ObjId, str, Metric], operator: str, values: Union[float, int, tuple[float, float]], treat_nulls_as: Union[float, None] = None)
```

Bases: *Filter*

```
__init__(metric: Union[ObjId, str, Metric], operator: str, values: Union[float, int, tuple[float, float]], treat_nulls_as: Union[float, None] = None) → None
```

Methods

`__init__(metric, operator, values[, ...])`

`as_api_model()`

`is_noop()`

Attributes

`apply_on_result`

`metric`

`operator`

`treat_nulls_as`

`values`

`gooddata_sdk.compute.model.filter.NegativeAttributeFilter`

`class gooddata_sdk.compute.model.filter.NegativeAttributeFilter`(*label: Union[ObjId, str, Attribute]*, *values: list[str] = None*)

Bases: *AttributeFilter*

`__init__(label: Union[ObjId, str, Attribute], values: list[str] = None) → None`

Methods

`__init__(label[, values])`

`as_api_model()`

`is_noop()`

Attributes

apply_on_result

label

values

gooddata_sdk.compute.model.filter.PositiveAttributeFilter

```
class gooddata_sdk.compute.model.filter.PositiveAttributeFilter(label: Union[ObjId, str,
                                                                    Attribute], values: list[str] =
                                                                    None)
```

Bases: *AttributeFilter*

```
__init__(label: Union[ObjId, str, Attribute], values: list[str] = None) → None
```

Methods

```
__init__(label[, values])
```

```
as_api_model()
```

```
is_noop()
```

Attributes

apply_on_result

label

values

gooddata_sdk.compute.model.filter.RankingFilter

```
class gooddata_sdk.compute.model.filter.RankingFilter(metrics: list[Union[ObjId, Metric, str]],
                                                       operator: str, value: int, dimensionality:
                                                       Optional[list[Union[str, ObjId, Attribute,
                                                                           Metric]]])
```

Bases: *Filter*

```
__init__(metrics: list[Union[ObjId, Metric, str]], operator: str, value: int, dimensionality:
         Optional[list[Union[str, ObjId, Attribute, Metric]]]) → None
```

Methods

`__init__(metrics, operator, value, ...)`

`as_api_model()`

`is_noop()`

Attributes

`apply_on_result`

`dimensionality`

`metrics`

`operator`

`value`

`gooddata_sdk.compute.model.filter.RelativeDateFilter`

`class gooddata_sdk.compute.model.filter.RelativeDateFilter(dataset: ObjId, granularity: str, from_shift: int, to_shift: int)`

Bases: `Filter`

`__init__(dataset: ObjId, granularity: str, from_shift: int, to_shift: int) → None`

Methods

`__init__(dataset, granularity, from_shift, ...)`

`as_api_model()`

`is_noop()`

Attributes

`apply_on_result`

`dataset`

`from_shift`

`granularity`

`to_shift`

gooddata_sdk.compute.model.metric**Classes**

`ArithmeticMetric`(local_id, operator, operands)

`Metric`(local_id)

`PopDate`(attribute, periods_ago)

`PopDateDataset`(dataset, periods_ago)

`PopDateMetric`(local_id, metric, date_attributes)

`PopDateSetMetric`(local_id, metric, date_datasets)

`SimpleMetric`(local_id, item[, aggregation, ...])

gooddata_sdk.compute.model.metric.ArithmeticMetric

```
class gooddata_sdk.compute.model.metric.ArithmeticMetric(local_id: str, operator: str, operands:
list[Union[str, Metric]])
```

```
    Bases: Metric
```

```
    __init__(local_id: str, operator: str, operands: list[Union[str, Metric]]) → None
```

Methods

`__init__(local_id, operator, operands)`

`as_api_model()`

Attributes

`local_id`

`operand_local_ids`

`operator`

`gooddata_sdk.compute.model.metric.Metric`

class `gooddata_sdk.compute.model.metric.Metric(local_id: str)`

Bases: `ExecModelEntity`

`__init__(local_id: str) → None`

Methods

`__init__(local_id)`

`as_api_model()`

Attributes

`local_id`

`gooddata_sdk.compute.model.metric.PopDate`

class `gooddata_sdk.compute.model.metric.PopDate(attribute: Union[ObjId, Attribute], periods_ago: int)`

Bases: `object`

`__init__(attribute: Union[ObjId, Attribute], periods_ago: int) → None`

Methods

`__init__(attribute, periods_ago)`

`as_api_model()`

Attributes

`attribute`

`periods_ago`

gooddata_sdk.compute.model.metric.PopDateDataset

```
class gooddata_sdk.compute.model.metric.PopDateDataset(dataset: Union[ObjId, str], periods_ago: int)
```

Bases: object

```
__init__(dataset: Union[ObjId, str], periods_ago: int) → None
```

Methods

`__init__(dataset, periods_ago)`

`as_api_model()`

Attributes

`dataset`

`periods_ago`

gooddata_sdk.compute.model.metric.PopDateMetric

```
class gooddata_sdk.compute.model.metric.PopDateMetric(local_id: str, metric: Union[str, Metric], date_attributes: list[PopDate])
```

Bases: *Metric*

```
__init__(local_id: str, metric: Union[str, Metric], date_attributes: list[PopDate]) → None
```

Methods

`__init__(local_id, metric, date_attributes)`

`as_api_model()`

Attributes

`date_attributes`

`local_id`

`metric_local_id`

`gooddata_sdk.compute.model.metric.PopDatasetMetric`

`class gooddata_sdk.compute.model.metric.PopDatasetMetric(local_id: str, metric: Union[str, Metric], date_datasets: list[PopDateDataset])`

Bases: `Metric`

`__init__(local_id: str, metric: Union[str, Metric], date_datasets: list[PopDateDataset])` → None

Methods

`__init__(local_id, metric, date_datasets)`

`as_api_model()`

Attributes

`date_datasets`

`local_id`

`metric_local_id`

gooddata_sdk.compute.model.metric.SimpleMetric

```
class gooddata_sdk.compute.model.metric.SimpleMetric(local_id: str, item: ObjId, aggregation:
Optional[str] = None, compute_ratio: bool =
False, filters: list[Filter] = None)
```

Bases: *Metric*

```
__init__(local_id: str, item: ObjId, aggregation: Optional[str] = None, compute_ratio: bool = False,
filters: list[Filter] = None) → None
```

Methods

```
__init__(local_id, item[, aggregation, ...])
```

```
as_api_model()
```

Attributes

```
aggregation
```

```
compute_ratio
```

```
filters
```

```
item
```

```
local_id
```

gooddata_sdk.compute.service**Classes**

<i>ComputeService</i> (api_client)	Compute service drives computation of analytics for a GoodData.CN workspaces.
------------------------------------	---

gooddata_sdk.compute.service.ComputeService

```
class gooddata_sdk.compute.service.ComputeService(api_client: GoodDataApiClient)
```

Bases: object

Compute service drives computation of analytics for a GoodData.CN workspaces. The prescription of what to compute is encapsulated by the ExecutionDefinition which consists of attributes, metrics, filters and definition of dimensions that influence how to organize the data in the result.

```
__init__(api_client: GoodDataApiClient)
```

Methods

<code>__init__(api_client)</code>	
<code>for_exec_def(workspace_id, exec_def)</code>	Starts computation in GoodData.CN workspace, using the provided execution definition.
<code>get_exec_metadata(workspace_id, result_id)</code>	Gets execution result's metadata from GoodData.CN workspace for given execution result ID.

for_exec_def(*workspace_id: str, exec_def: ExecutionDefinition*) → *Execution*

Starts computation in GoodData.CN workspace, using the provided execution definition.

Parameters

- **workspace_id** – workspace identifier
- **exec_def** – execution definition - this prescribes what to calculate, how to place labels and metric values into dimensions

get_exec_metadata(*workspace_id: str, result_id: str*) → *ResultCacheMetadata*

Gets execution result's metadata from GoodData.CN workspace for given execution result ID.

Parameters

- **workspace_id** – workspace identifier
- **result_id** – execution result ID

Returns

execution result's metadata

3.1.4 gooddata_sdk.insight

Classes

<code>Insight(from_vis_obj[, side_loads])</code>	
<code>InsightAttribute(attribute)</code>	
<code>InsightBucket(bucket)</code>	
<code>InsightFilter(f)</code>	
<code>InsightMetric(metric)</code>	Represents metric placed on an insight.
<code>InsightService(api_client)</code>	Insight Service allows retrieval of insights from a GD.CN workspace.

gooddata_sdk.insight.Insight

```
class gooddata_sdk.insight.Insight(from_vis_obj: dict[str, Any], side_loads: Optional[SideLoads] = None)
```

Bases: object

```
__init__(from_vis_obj: dict[str, Any], side_loads: Optional[SideLoads] = None) → None
```

Methods

```
__init__(from_vis_obj[, side_loads])
```

```
get_metadata(id_obj)
```

Attributes

```
are_relations_valid
```

```
attributes
```

```
buckets
```

```
description
```

```
filters
```

```
id
```

```
metrics
```

```
properties
```

```
side_loads
```

```
sorts
```

```
title
```

```
vis_url
```

gooddata_sdk.insight.InsightAttribute

class gooddata_sdk.insight.InsightAttribute(*attribute: dict[str, Any]*)

Bases: object

`__init__`(*attribute: dict[str, Any]*) → None

Methods

`__init__`(*attribute*)

`as_computable`()

Attributes

`alias`

`label`

`label_id`

`local_id`

gooddata_sdk.insight.InsightBucket

class gooddata_sdk.insight.InsightBucket(*bucket: dict[str, Any]*)

Bases: object

`__init__`(*bucket: dict[str, Any]*) → None

Methods

`__init__`(*bucket*)

Attributes

`attributes`

`items`

`local_id`

`metrics`

gooddata_sdk.insight.InsightFilter

```
class gooddata_sdk.insight.InsightFilter(f: dict[str, Any])
```

Bases: object

```
__init__(f: dict[str, Any]) → None
```

Methods

```
__init__(f)
```

```
as_computable()
```

gooddata_sdk.insight.InsightMetric

```
class gooddata_sdk.insight.InsightMetric(metric: dict[str, Any])
```

Bases: object

Represents metric placed on an insight.

Note: this has different shape than object passed to execution.

```
__init__(metric: dict[str, Any]) → None
```

Methods

```
__init__(metric)
```

```
as_computable()
```

Attributes

```
alias
```

```
format
```

```
is_time_comparison
```

```
item
```

```
item_id
```

```
local_id
```

```
time_comparison_master
```

If this is a time comparison metric, return local_id of the master metric from which it is derived.

```
title
```

property time_comparison_master: Optional[str]

If this is a time comparison metric, return local_id of the master metric from which it is derived.

Returns

local_id of master metric, None if not a time comparison metric

gooddata_sdk.insight.InsightService

class gooddata_sdk.insight.InsightService(*api_client*: GoodDataApiClient)

Bases: object

Insight Service allows retrieval of insights from a GD.CN workspace. The insights are returned as instances of Insight which allows convenient introspection and necessary functions to convert the insight into a form where it can be sent for computation.

Note: the insights are created using GD.CN Analytical Designer or using GoodData.UI SDK. They are stored as visualization objects with a free-form body. This body is specific for AD & SDK. The Insight wrapper exists to take care of these discrepancies.

__init__(*api_client*: GoodDataApiClient) → None

Methods

<code>__init__(api_client)</code>	
<code>get_insight(workspace_id, insight_id)</code>	Gets a single insight from a workspace.
<code>get_insights(workspace_id)</code>	Gets all insights for a workspace.

get_insight(*workspace_id*: str, *insight_id*: str) → *Insight*

Gets a single insight from a workspace.

Parameters

- **workspace_id** – identifier of workspace to load insight from
- **insight_id** – identifier of the insight

Returns

single insight; the insight will contain sideloaded metadata about the entities it references

Return type

Insight

get_insights(*workspace_id*: str) → list[*Insight*]

Gets all insights for a workspace. The insights will contain side loaded metadata for all execution entities that they reference.

Parameters

workspace_id – identifier of workspace to load insights from

Returns

all available insights, each insight will contain side loaded metadata about the entities it references

3.1.5 gooddata_sdk.sdk

Classes

<code>GoodDataSdk(client)</code>	Top-level class that wraps all the functionality together.
----------------------------------	--

gooddata_sdk.sdk.GoodDataSdk

class gooddata_sdk.sdk.GoodDataSdk(*client*: GoodDataApiClient)

Bases: object

Top-level class that wraps all the functionality together.

`__init__(client: GoodDataApiClient) → None`

Take instance of GoodDataApiClient and return new GoodDataSdk instance.

Useful when customized GoodDataApiClient is needed. Usually users should use `GoodDataSdk.create` classmethod.

Methods

<code>__init__(client)</code>	Take instance of GoodDataApiClient and return new GoodDataSdk instance.
<code>create(host_, token_[, extra_user_agent_])</code>	Create common GoodDataApiClient and return new GoodDataSdk instance.

Attributes

<code>catalog_data_source</code>
<code>catalog_organization</code>
<code>catalog_permission</code>
<code>catalog_user</code>
<code>catalog_workspace</code>
<code>catalog_workspace_content</code>
<code>client</code>
<code>compute</code>
<code>insights</code>
<code>support</code>
<code>tables</code>

```
classmethod create(host_: str, token_: str, extra_user_agent_: Optional[str] = None,  
                    **custom_headers_: Optional[str]) → GoodDataSdk
```

Create common GoodDataApiClient and return new GoodDataSdk instance. Custom headers are filtered. Headers with None value are removed. It simplifies usage because headers can be created directly from optional values.

This is preferred way of creating GoodDataSdk, when no tweaks are needed.

3.1.6 gooddata_sdk.support

Classes

SupportService(api_client)

gooddata_sdk.support.SupportService

```
class gooddata_sdk.support.SupportService(api_client: GoodDataApiClient)
```

Bases: object

```
__init__(api_client: GoodDataApiClient) → None
```

Methods

```
__init__(api_client)
```

```
wait_till_available(timeout[, sleep_time])      Wait till GD.CN service is available.
```

Attributes

```
is_available      Checks if GD.CN is available.
```

```
property is_available: bool
```

Checks if GD.CN is available. Can raise exceptions in case of authentication or authorization failure.

Returns

True - available, False - not available

```
wait_till_available(timeout: int, sleep_time: float = 2.0) → None
```

Wait till GD.CN service is available. When timeout is:

- > 0 exception is raised after given number of seconds.
- = 0 exception is raised whe service is not available immediately
- < 0 no timeout

Method propagates is_available exceptions.

Parameters

- **timeout** – seconds to wait to service to be available (see method description for details)
- **sleep_time** – seconds to wait between GD.CN availability tests

3.1.7 gooddata_sdk.table

Classes

<code>ExecutionTable(response, first_page)</code>	Represents execution result as a table.
<code>TableService(api_client)</code>	The TableService provides a convenient way to drive computations and access the results in a tabular fashion.

gooddata_sdk.table.ExecutionTable

class `gooddata_sdk.table.ExecutionTable`(*response*: `Execution`, *first_page*: `ExecutionResult`)

Bases: `object`

Represents execution result as a table. This is a convenience wrapper for executions constructed using the following convention:

- all attributes are in the first dimension
- all metrics are in the second dimension
- if the execution is attribute- or metric-less, then there is always single dimension

The mapping to rows is then as follows:

- both attributes + metrics are on the execution = iteration over first dimension; as many rows as total records in the first dimension (`paging.total[0]`)
- just attributes = iteration over just headers in first dimension; as many rows as total records in the first dimension (`paging.total[0]`)
- just metrics = single row, all metrics values returned in one row

`__init__`(*response*: `Execution`, *first_page*: `ExecutionResult`) → `None`

Methods

<code>__init__(response, first_page)</code>	
<code>read_all()</code>	Returns a generator that will be yielding execution result as rows.

Attributes

attributes	
<i>column_ids</i>	Returns column identifiers.
<i>column_metadata</i>	Returns mapping of column identifier to definition of either attribute whose elements will be in that column or metric whose value will be calculated in that column.
metrics	

property column_ids: list[str]

Returns column identifiers. Each row will be a mapping of column identifier to column data.

property column_metadata: dict[str, Union[Attribute, Metric]]

Returns mapping of column identifier to definition of either attribute whose elements will be in that column or metric whose value will be calculated in that column.

read_all() → Generator[dict[str, Any], None, None]

Returns a generator that will be yielding execution result as rows. Each row is a dict() mapping column identifier to value of that column.

Returns

generator yielding dict() representing rows of the table

gooddata_sdk.table.TableService

class gooddata_sdk.table.TableService(api_client: GoodDataApiClient)

Bases: object

The TableService provides a convenient way to drive computations and access the results in a tabular fashion.

Compared to the ComputeService, with this one here you do not have to worry about the layout of the result and do not have to have to work with execution response, access the data using paging.

The ExecutionTable returned by the TableService allows you to iterate over the rows of the calculated data.

__init__(api_client: GoodDataApiClient) → None

Methods

__init__(api_client)

for_insight(workspace_id, insight)

for_items(workspace_id, items[, filters])

3.1.8 gooddata_sdk.type_converter

Functions

<i>build_stores()</i>	Initialize both AttributeConverterStore and DBTypeConverterStore with Convertors.
-----------------------	---

gooddata_sdk.type_converter.build_stores

gooddata_sdk.type_converter.build_stores() → None

Initialize both AttributeConverterStore and DBTypeConverterStore with Convertors.

Classes

<i>AttributeConverterStore()</i>	Store for conversion of attributes
<i>Converter()</i>	Base Converter class.
<i>ConverterRegistryStore()</i>	Class store TypeConverterRegistry instances for each registered type.
<i>DBTypeConverterStore()</i>	Store for conversion of database types
<i>DateConverter()</i>	
<i>DatetimeConverter()</i>	
<i>IntegerConverter()</i>	
<i>StringConverter()</i>	
<i>TypeConverterRegistry(type_name)</i>	Class stores converters for given type with ability to distinguish converters based on sub-type granularity.

gooddata_sdk.type_converter.AttributeConverterStore

class gooddata_sdk.type_converter.AttributeConverterStore

Bases: *ConverterRegistryStore*

Store for conversion of attributes

__init__()

Methods

<i>__init__()</i>	
<i>find_converter(type_name[, sub_type])</i>	Find Converter for given type and sub type.
<i>register(type_name, class_converter[, sub_types])</i>	Register Converter instance created from provided Converter class to given type and list of sub types.
<i>reset()</i>	Reset converters setup

classmethod `find_converter`(*type_name: str, sub_type: Optional[str] = None*) → *Converter*

Find Converter for given type and sub type.

Parameters

- **type_name** – type name
- **sub_type** – sub type name

classmethod `register`(*type_name: str, class_converter: Type[Converter], sub_types: Optional[list[str]] = None*) → *None*

Register Converter instance created from provided Converter class to given type and list of sub types. When sub types are not provided, converter is registered as the default one for given type.

Parameters

- **type_name** – type name
- **class_converter** – Converter class
- **sub_types** – list of sub types or None (default type Converter)

classmethod `reset`() → *None*

Reset converters setup

gooddata_sdk.type_converter.Converter

class `gooddata_sdk.type_converter.Converter`

Bases: `object`

Base Converter class. It defines Converter API and implements support for external type conversion. External type conversion provides ability to plug-in conversion function to Converter

`__init__()`

Methods

`__init__()`

`db_data_type()`

`set_external_fnc(fnc)`

`to_external_type(value)`

`to_type(value)`

Attributes

DEFAULT_DB_DATA_TYPE

gooddata_sdk.type_converter.ConverterRegistryStore

class gooddata_sdk.type_converter.ConverterRegistryStore

Bases: object

Class store TypeConverterRegistry instances for each registered type. It provides interface to register converters with type and sub-type and to find converter. The class is not meant to be used directly but as base class for child classes

`__init__()`

Methods

`__init__()`

<code>find_converter(type_name[, sub_type])</code>	Find Converter for given type and sub type.
<code>register(type_name, class_converter[, sub_types])</code>	Register Converter instance created from provided Converter class to given type and list of sub types.
<code>reset()</code>	Reset converters setup

classmethod `find_converter`(*type_name*: str, *sub_type*: Optional[str] = None) → Converter

Find Converter for given type and sub type.

Parameters

- **type_name** – type name
- **sub_type** – sub type name

classmethod `register`(*type_name*: str, *class_converter*: Type[Converter], *sub_types*: Optional[list[str]] = None) → None

Register Converter instance created from provided Converter class to given type and list of sub types. When sub types are not provided, converter is registered as the default one for given type.

Parameters

- **type_name** – type name
- **class_converter** – Converter class
- **sub_types** – list of sub types or None (default type Converter)

classmethod `reset`() → None

Reset converters setup

gooddata_sdk.type_converter.DBTypeConverterStore**class** gooddata_sdk.type_converter.DBTypeConverterStoreBases: *ConverterRegistryStore*

Store for conversion of database types

__init__()**Methods**

<i>__init__</i> ()	
<i>find_converter</i> (type_name[, sub_type])	Find Converter for given type and sub type.
<i>register</i> (type_name, class_converter[, sub_types])	Register Converter instance created from provided Converter class to given type and list of sub types.
<i>reset</i> ()	Reset converters setup

classmethod **find_converter**(type_name: str, sub_type: Optional[str] = None) → *Converter*

Find Converter for given type and sub type.

Parameters

- **type_name** – type name
- **sub_type** – sub type name

classmethod **register**(type_name: str, class_converter: Type[*Converter*], sub_types: Optional[list[str]] = None) → None

Register Converter instance created from provided Converter class to given type and list of sub types. When sub types are not provided, converter is registered as the default one for given type.

Parameters

- **type_name** – type name
- **class_converter** – Converter class
- **sub_types** – list of sub types or None (default type Converter)

classmethod **reset**() → None

Reset converters setup

gooddata_sdk.type_converter.DateConverter**class** gooddata_sdk.type_converter.DateConverterBases: *Converter***__init__**()

Methods

<code>__init__()</code>	
<code>db_data_type()</code>	
<code>set_external_fnc(fnc)</code>	
<code>to_date(value)</code>	Add first month and first date to incomplete iso date string.
<code>to_external_type(value)</code>	
<code>to_type(value)</code>	

Attributes

<code>DEFAULT_DB_DATA_TYPE</code>

classmethod `to_date(value: str) → date`

Add first month and first date to incomplete iso date string.

```
>>> assert DateConverter.to_date("2021-01") == date(2021, 1, 1)
>>> assert DateConverter.to_date("1992") == date(1992, 1, 1)
```

`gooddata_sdk.type_converter.DatetimeConverter`

class `gooddata_sdk.type_converter.DatetimeConverter`

Bases: `Converter`

`__init__()`

Methods

<code>__init__()</code>	
<code>db_data_type()</code>	
<code>set_external_fnc(fnc)</code>	
<code>to_datetime(value)</code>	Append minutes to incomplete datetime string.
<code>to_external_type(value)</code>	
<code>to_type(value)</code>	

Attributes

DEFAULT_DB_DATA_TYPE

classmethod `to_datetime`(*value: str*) → datetime

Append minutes to incomplete datetime string.

```
>>> from datetime import datetime
>>> assert DatetimeConverter.to_datetime("2021-01-01 02") == datetime(2021, 1,
↳1, 2, 0)
>>> assert DatetimeConverter.to_datetime("2021-01-01 12:34") == datetime(2021,
↳1, 1, 12, 34)
```

`gooddata_sdk.type_converter.IntegerConverter`

class `gooddata_sdk.type_converter.IntegerConverter`

Bases: `Converter`

`__init__()`

Methods

`__init__()`

`db_data_type()`

`set_external_fnc(fnc)`

`to_external_type(value)`

`to_type(value)`

Attributes

DEFAULT_DB_DATA_TYPE

gooddata_sdk.type_converter.StringConverter**class** gooddata_sdk.type_converter.StringConverterBases: *Converter*`__init__()`**Methods**`__init__()``db_data_type()``set_external_fnc(fnc)``to_external_type(value)``to_type(value)`**Attributes**`DEFAULT_DB_DATA_TYPE`**gooddata_sdk.type_converter.TypeConverterRegistry****class** gooddata_sdk.type_converter.TypeConverterRegistry(*type_name: str*)Bases: `object`

Class stores converters for given type with ability to distinguish converters based on sub-type granularity.

`__init__(type_name: str)`

Initialize instance with type for which instance is going to be responsible

Parameters**type_name** – type name**Methods**`__init__(type_name)`

Initialize instance with type for which instance is going to be responsible

`converter(sub_type)`

Find and return converter instance for a given sub-type.

`register(converter, sub_type)`

Register converter instance for given sub-type (granularity).

converter(*sub_type: Optional[str]*) → *Converter*

Find and return converter instance for a given sub-type. Default converter instance is returned if the sub-type is not found or not provided. When a default converter is not registered, ValueError exception is raised.

Parameters

sub_type – sub-type name

Returns

Converter instance

register(*converter: Converter, sub_type: Optional[str]*) → None

Register converter instance for given sub-type (granularity). If sub-type is not specified, converter is registered as the default one for the whole type. Default converter can be registered only once.

Parameters

- **converter** – converter instance
- **sub_type** – sub-type name

3.1.9 gooddata_sdk.utils

Functions

camel_to_snake(camel_case_str)

change_case(dictionary, case)

change_case_helper(value, case)

create_directory(path)

get_sorted_yaml_files(folder)

id_obj_to_key(id_obj)

Given an object containing an id+type pair, this function will return a string key.

load_all_entities(get_page_func[, page_size])

Loads all entities from a paged resource.

load_all_entities_dict(get_page_func[, ...])

read_layout_from_file(path)

snake_to_camel(snake_case_str)

write_layout_to_file(path, content)

gooddata_sdk.utils.camel_to_snake

`gooddata_sdk.utils.camel_to_snake(camel_case_str: str) → str`

gooddata_sdk.utils.change_case

`gooddata_sdk.utils.change_case(dictionary: dict, case: Callable[[str], str]) → dict`

gooddata_sdk.utils.change_case_helper

`gooddata_sdk.utils.change_case_helper(value: Union[list, dict, str], case: Callable[[str], str]) → Union[list, dict, str]`

gooddata_sdk.utils.create_directory

`gooddata_sdk.utils.create_directory(path: Path) → None`

gooddata_sdk.utils.get_sorted_yaml_files

`gooddata_sdk.utils.get_sorted_yaml_files(folder: Path) → list[Path]`

gooddata_sdk.utils.id_obj_to_key

`gooddata_sdk.utils.id_obj_to_key(id_obj: Union[str, ObjId, Dict[str, Dict[str, str]], Dict[str, str]]) → str`

Given an object containing an id+type pair, this function will return a string key.

For convenience, this also recognizes the *ref* format used by GoodData.UI SDK. In that format, the id+type are wrapped in 'identifier'.

Parameters

id_obj – id object

Returns

string that can be used as key

gooddata_sdk.utils.load_all_entities

`gooddata_sdk.utils.load_all_entities(get_page_func: functools.partial[Any], page_size: int = 500) → AllPagedEntities`

Loads all entities from a paged resource. The primary input to this function is a partial function that is setup with all the fixed parameters. Given this the function will get entities page-by-page and merge them into a single 'pseudo-response' containing data and included attributes.

An example usage:

```
>>> import functools
>>> import gooddata_metadata_client as metadata_client
>>> import gooddata_metadata_client.apis as metadata_apis
>>> api = metadata_apis.EntitiesApi(metadata_client.ApiClient())
```

(continues on next page)

(continued from previous page)

```

>>> get_func = functools.partial(api.get_all_entities_visualization_objects, 'some-
↳workspace-id',
>>>                                     include=["ALL"], _check_return_type=False)
>>> vis_objects = load_all_entities(get_func)

```

Parameters

- **get_page_func** – an API controller from the metadata client
- **page_size** – optionally specify page length, default is 500

gooddata_sdk.utils.load_all_entities_dict

```
gooddata_sdk.utils.load_all_entities_dict(get_page_func: functools.partial[Any], page_size: int = 500,
                                          camel_case: bool = False) → dict[str, Any]
```

gooddata_sdk.utils.read_layout_from_file

```
gooddata_sdk.utils.read_layout_from_file(path: Path) → Any
```

gooddata_sdk.utils.snake_to_camel

```
gooddata_sdk.utils.snake_to_camel(snake_case_str: str) → str
```

gooddata_sdk.utils.write_layout_to_file

```
gooddata_sdk.utils.write_layout_to_file(path: Path, content: Union[dict[str, Any], list[dict]]) → None
```

Classes

```
AllPagedEntities(data, included)
```

```
IndentDumper(stream[, default_style, ...])
```

```
SideLoads(objs)
```

gooddata_sdk.utils.AllPagedEntities

```
class gooddata_sdk.utils.AllPagedEntities(data, included)
```

```
    Bases: tuple
```

```
    __init__()
```

Methods

<code>__init__()</code>	
<code>count(value, /)</code>	Return number of occurrences of value.
<code>index(value[, start, stop])</code>	Return first index of value.

Attributes

<code>data</code>	Alias for field number 0
<code>included</code>	Alias for field number 1

count(*value*, /)

Return number of occurrences of value.

property data

Alias for field number 0

property included

Alias for field number 1

index(*value*, *start*=0, *stop*=9223372036854775807, /)

Return first index of value.

Raises ValueError if the value is not present.

gooddata_sdk.utils.IndentDumper

class gooddata_sdk.utils.**IndentDumper**(*stream*, *default_style*=None, *default_flow_style*=False, *canonical*=None, *indent*=None, *width*=None, *allow_unicode*=None, *line_break*=None, *encoding*=None, *explicit_start*=None, *explicit_end*=None, *version*=None, *tags*=None, *sort_keys*=True)

Bases: SafeDumper

__init__(*stream*, *default_style*=None, *default_flow_style*=False, *canonical*=None, *indent*=None, *width*=None, *allow_unicode*=None, *line_break*=None, *encoding*=None, *explicit_start*=None, *explicit_end*=None, *version*=None, *tags*=None, *sort_keys*=True)

Methods

<code>__init__(stream[, default_style, ...])</code>
<code>add_implicit_resolver(tag, regexp, first)</code>
<code>add_multi_representer(data_type, representer)</code>
<code>add_path_resolver(tag, path[, kind])</code>

continues on next page

Table 1 – continued from previous page

<code>add_representer(data_type, representer)</code>
<code>analyze_scalar(scalar)</code>
<code>anchor_node(node)</code>
<code>ascend_resolver()</code>
<code>check_empty_document()</code>
<code>check_empty_mapping()</code>
<code>check_empty_sequence()</code>
<code>check_resolver_prefix(depth, path, kind, ...)</code>
<code>check_simple_key()</code>
<code>choose_scalar_style()</code>
<code>close()</code>
<code>descend_resolver(current_node, current_index)</code>
<code>determine_block_hints(text)</code>
<code>dispose()</code>
<code>emit(event)</code>
<code>expect_alias()</code>
<code>expect_block_mapping()</code>
<code>expect_block_mapping_key([first])</code>
<code>expect_block_mapping_simple_value()</code>
<code>expect_block_mapping_value()</code>
<code>expect_block_sequence()</code>
<code>expect_block_sequence_item([first])</code>
<code>expect_document_end()</code>
<code>expect_document_root()</code>
<code>expect_document_start([first])</code>

continues on next page

Table 1 – continued from previous page

<code>expect_first_block_mapping_key()</code>
<code>expect_first_block_sequence_item()</code>
<code>expect_first_document_start()</code>
<code>expect_first_flow_mapping_key()</code>
<code>expect_first_flow_sequence_item()</code>
<code>expect_flow_mapping()</code>
<code>expect_flow_mapping_key()</code>
<code>expect_flow_mapping_simple_value()</code>
<code>expect_flow_mapping_value()</code>
<code>expect_flow_sequence()</code>
<code>expect_flow_sequence_item()</code>
<code>expect_node([root, sequence, mapping, ...])</code>
<code>expect_nothing()</code>
<code>expect_scalar()</code>
<code>expect_stream_start()</code>
<code>flush_stream()</code>
<code>generate_anchor(node)</code>
<code>ignore_aliases(data)</code>
<code>increase_indent([flow, indentless])</code>
<code>need_events(count)</code>
<code>need_more_events()</code>
<code>open()</code>
<code>prepare_anchor(anchor)</code>
<code>prepare_tag(tag)</code>
<code>prepare_tag_handle(handle)</code>

continues on next page

Table 1 – continued from previous page

<code>prepare_tag_prefix(prefix)</code>
<code>prepare_version(version)</code>
<code>process_anchor(indicator)</code>
<code>process_scalar()</code>
<code>process_tag()</code>
<code>represent(data)</code>
<code>represent_binary(data)</code>
<code>represent_bool(data)</code>
<code>represent_data(data)</code>
<code>represent_date(data)</code>
<code>represent_datetime(data)</code>
<code>represent_dict(data)</code>
<code>represent_float(data)</code>
<code>represent_int(data)</code>
<code>represent_list(data)</code>
<code>represent_mapping(tag, mapping[, flow_style])</code>
<code>represent_none(data)</code>
<code>represent_scalar(tag, value[, style])</code>
<code>represent_sequence(tag, sequence[, flow_style])</code>
<code>represent_set(data)</code>
<code>represent_str(data)</code>
<code>represent_undefined(data)</code>
<code>represent_yaml_object(tag, data, cls[, ...])</code>
<code>resolve(kind, value, implicit)</code>
<code>serialize(node)</code>

continues on next page

Table 1 – continued from previous page

<code>serialize_node(node, parent, index)</code>
<code>write_double_quoted(text[, split])</code>
<code>write_folded(text)</code>
<code>write_indent()</code>
<code>write_indicator(indicator, need_whitespace)</code>
<code>write_line_break([data])</code>
<code>write_literal(text)</code>
<code>write_plain(text[, split])</code>
<code>write_single_quoted(text[, split])</code>
<code>write_stream_end()</code>
<code>write_stream_start()</code>
<code>write_tag_directive(handle_text, prefix_text)</code>
<code>write_version_directive(version_text)</code>

Attributes

<code>ANCHOR_TEMPLATE</code>
<code>DEFAULT_MAPPING_TAG</code>
<code>DEFAULT_SCALAR_TAG</code>
<code>DEFAULT_SEQUENCE_TAG</code>
<code>DEFAULT_TAG_PREFIXES</code>
<code>ESCAPE_REPLACEMENTS</code>
<code>inf_value</code>
<code>yaml_implicit_resolvers</code>
<code>yaml_multi_representers</code>
<code>yaml_path_resolvers</code>
<code>yaml_representers</code>

gooddata_sdk.utils.SideLoads

class gooddata_sdk.utils.SideLoads(*objs: list[Any]*)

Bases: object

__init__(*objs: list[Any]*) → None

Methods

__init__(*objs*)

all_for_type(*obj_type*)

find(*id_obj*)

PYTHON MODULE INDEX

g

gooddata_sdk, 31

gooddata_sdk.catalog, 32

gooddata_sdk.catalog.base, 32

gooddata_sdk.catalog.catalog_service_base, 34

gooddata_sdk.catalog.data_source, 34

gooddata_sdk.catalog.data_source.action_requests, 35

gooddata_sdk.catalog.data_source.action_requests.idm_request, 35

gooddata_sdk.catalog.data_source.action_requests.scan_model_request, 38

gooddata_sdk.catalog.data_source.declarative_model, 40

gooddata_sdk.catalog.data_source.declarative_model.data_source, 41

gooddata_sdk.catalog.data_source.declarative_model.physical_model, 44

gooddata_sdk.catalog.data_source.declarative_model.physical_model.column, 45

gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm, 46

gooddata_sdk.catalog.data_source.declarative_model.physical_model.table, 49

gooddata_sdk.catalog.data_source.entity_model, 51

gooddata_sdk.catalog.data_source.entity_model.content_objects, 51

gooddata_sdk.catalog.data_source.entity_model.content_objects.table, 51

gooddata_sdk.catalog.data_source.entity_model.data_source, 56

gooddata_sdk.catalog.data_source.service, 70

gooddata_sdk.catalog.data_source.validation, 73

gooddata_sdk.catalog.data_source.validation.data_source, 73

gooddata_sdk.catalog.entity, 74

gooddata_sdk.catalog.identifier, 78

gooddata_sdk.catalog.organization, 84

gooddata_sdk.catalog.organization.entity_model, 84

gooddata_sdk.catalog.organization.entity_model.organization, 84

gooddata_sdk.catalog.organization.service, 88

gooddata_sdk.catalog.permission, 89

gooddata_sdk.catalog.permission.declarative_model, 89

gooddata_sdk.catalog.permission.declarative_model.permission, 89

gooddata_sdk.catalog.permission.service, 94

gooddata_sdk.catalog.setting, 95

gooddata_sdk.catalog.types, 96

gooddata_sdk.catalog.user, 96

gooddata_sdk.catalog.user.declarative_model, 96

gooddata_sdk.catalog.user.declarative_model.user, 96

gooddata_sdk.catalog.user.declarative_model.user_and_user_group, 98

gooddata_sdk.catalog.user.declarative_model.user_group, 99

gooddata_sdk.catalog.user.entity_model, 102

gooddata_sdk.catalog.user.entity_model.user, 102

gooddata_sdk.catalog.user.entity_model.user_group, 107

gooddata_sdk.catalog.user.service, 111

gooddata_sdk.catalog.workspace, 114

gooddata_sdk.catalog.workspace.content_service, 114

gooddata_sdk.catalog.workspace.declarative_model, 116

gooddata_sdk.catalog.workspace.declarative_model.workspace, 116

gooddata_sdk.catalog.workspace.declarative_model.workspace.declarative_model, 117

gooddata_sdk.catalog.workspace.declarative_model.workspace.declarative_model.workspace, 117

gooddata_sdk.catalog.workspace.declarative_model.workspace.declarative_model.workspace.declarative_model, 129

gooddata_sdk.catalog.workspace.declarative_model.workspace.declarative_model.workspace.declarative_model.workspace, 130

gooddata_sdk.catalog.workspace.declarative_model.workspace.declarative_model.workspace.declarative_model.workspace.declarative_model, 130

- 130
- gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset,
140
- gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset,
140
- gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm,
144
- gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace,
146
- gooddata_sdk.catalog.workspace.entity_model,
155
- gooddata_sdk.catalog.workspace.entity_model.content_objects,
156
- gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset,
156
- gooddata_sdk.catalog.workspace.entity_model.content_objects.metric,
160
- gooddata_sdk.catalog.workspace.entity_model.graph_objects,
161
- gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph,
161
- gooddata_sdk.catalog.workspace.entity_model.workspace,
166
- gooddata_sdk.catalog.workspace.model_container,
167
- gooddata_sdk.catalog.workspace.service, 169
- gooddata_sdk.client, 171
- gooddata_sdk.compute, 172
- gooddata_sdk.compute.model, 172
- gooddata_sdk.compute.model.attribute, 173
- gooddata_sdk.compute.model.base, 173
- gooddata_sdk.compute.model.execution, 175
- gooddata_sdk.compute.model.filter, 181
- gooddata_sdk.compute.model.metric, 187
- gooddata_sdk.compute.service, 191
- gooddata_sdk.insight, 192
- gooddata_sdk.sdk, 197
- gooddata_sdk.support, 198
- gooddata_sdk.table, 199
- gooddata_sdk.type_converter, 201
- gooddata_sdk.utils, 208

INDEX

Symbols

- `__init__` () (*gooddata_sdk.catalog.base.Base* method), 33
- `__init__` () (*gooddata_sdk.catalog.catalog_service_base.CatalogServiceBase* method), 34
- `__init__` () (*gooddata_sdk.catalog.data_source.action_request.ActionRequest* method), 37
- `__init__` () (*gooddata_sdk.catalog.data_source.action_request.ScanModelRequest* method), 39
- `__init__` () (*gooddata_sdk.catalog.data_source.declarative_model.data_source.service.CatalogDataSource* method), 42
- `__init__` () (*gooddata_sdk.catalog.data_source.declarative_model.data_source.validation.data_source.DataSourcees* method), 43
- `__init__` () (*gooddata_sdk.catalog.data_source.declarative_model.physical_model.column.CatalogDeclarativeColumn* method), 45
- `__init__` () (*gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.CatalogDeclarativeTable* method), 47
- `__init__` () (*gooddata_sdk.catalog.data_source.declarative_model.physical_model.pam.CatalogScanResultPam* method), 48
- `__init__` () (*gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.CatalogDeclarativeTableEntity* method), 50
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTable* method), 52
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTableAttributes* method), 53
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTableColumn* method), 54
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.BigQueryAttributes* method), 56
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceIdentifier.CatalogAssigneeIdentifier* method), 57
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceIdentifier.CatalogGrainIdentifier* method), 59
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceIdentifier.CatalogLabelIdentifier* method), 61
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceIdentifier.CatalogReferenceIdentifier* method), 63
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceIdentifier.CatalogUserGroupIdentifier* method), 65
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.CatalogDataSourceIdentifier.CatalogWorkspaceIdentifier* method), 67
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.DatabaseAttributes* method), 68
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.entity_model.data_source.CatalogDataSourceEntityModel* method), 68
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.CatalogDataSourceEntityModelDataSources* method), 69
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.CatalogGenerateTableRequest* method), 70
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.CatalogScanModelRequest* method), 70
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.CatalogDeclarativeDataSource* method), 71
- `__init__` () (*gooddata_sdk.catalog.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.entity_model.data_source.CatalogDeclarativeDataSourcees* method), 73
- `__init__` () (*gooddata_sdk.catalog.entity.BasicCredentials* method), 74
- `__init__` () (*gooddata_sdk.catalog.entity.CatalogEntity* method), 75
- `__init__` () (*gooddata_sdk.catalog.entity.CatalogNameEntity* method), 75
- `__init__` () (*gooddata_sdk.catalog.entity.CatalogTitleEntity* method), 76
- `__init__` () (*gooddata_sdk.catalog.entity.CatalogTypeEntity* method), 76
- `__init__` () (*gooddata_sdk.catalog.entity.Credentials* method), 76
- `__init__` () (*gooddata_sdk.catalog.entity.TokenCredentials* method), 77
- `__init__` () (*gooddata_sdk.catalog.entity.TokenCredentialsFromFile* method), 78
- `__init__` () (*gooddata_sdk.catalog.identifier.CatalogAssigneeIdentifier* method), 79
- `__init__` () (*gooddata_sdk.catalog.identifier.CatalogGrainIdentifier* method), 79
- `__init__` () (*gooddata_sdk.catalog.identifier.CatalogLabelIdentifier* method), 80
- `__init__` () (*gooddata_sdk.catalog.identifier.CatalogReferenceIdentifier* method), 81
- `__init__` () (*gooddata_sdk.catalog.identifier.CatalogUserGroupIdentifier* method), 82
- `__init__` () (*gooddata_sdk.catalog.identifier.CatalogWorkspaceIdentifier* method), 83
- `__init__` () (*gooddata_sdk.catalog.organization.entity_model.organization.DatabaseAttributes* method), 83

method), 160	method), 186
__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 161	__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 187
__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 162	__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 188
__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 163	__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 188
__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 164	__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 189
__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 165	__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 189
__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 166	__init__ (gooddata_sdk.catalog.workspace.entity_model. method), 190
__init__ (gooddata_sdk.catalog.workspace.model_containers. method), 167	__init__ (gooddata_sdk.catalog.workspace.model_containers. method), 191
__init__ (gooddata_sdk.catalog.workspace.service.CatalogWorkspaceService. method), 169	__init__ (gooddata_sdk.compute.service.ComputeService method), 191
__init__ (gooddata_sdk.client.GoodDataApiClient method), 171	__init__ (gooddata_sdk.insight.Insight method), 193
__init__ (gooddata_sdk.compute.model.attribute.Attribute method), 173	__init__ (gooddata_sdk.insight.InsightAttribute method), 194
__init__ (gooddata_sdk.compute.model.base.ExecModelEntity method), 174	__init__ (gooddata_sdk.insight.InsightBucket method), 194
__init__ (gooddata_sdk.compute.model.base.Filter method), 174	__init__ (gooddata_sdk.insight.InsightFilter method), 195
__init__ (gooddata_sdk.compute.model.base.ObjId method), 174	__init__ (gooddata_sdk.insight.InsightMetric method), 195
__init__ (gooddata_sdk.compute.model.execution.BareExecutionResponse. method), 176	__init__ (gooddata_sdk.insight.InsightService method), 196
__init__ (gooddata_sdk.compute.model.execution.Execution method), 177	__init__ (gooddata_sdk.sdk.GoodDataSdk method), 197
__init__ (gooddata_sdk.compute.model.execution.ExecutionDefinition. method), 177	__init__ (gooddata_sdk.support.SupportService method), 198
__init__ (gooddata_sdk.compute.model.execution.ExecutionResult. method), 178	__init__ (gooddata_sdk.table.ExecutionTable method), 199
__init__ (gooddata_sdk.compute.model.execution.TotalDefinition. method), 179	__init__ (gooddata_sdk.table.TableService method), 199
__init__ (gooddata_sdk.compute.model.execution.TotalDimension. method), 180	__init__ (gooddata_sdk.type_converter.AttributeConverterStore method), 201
__init__ (gooddata_sdk.compute.model.filter.AbsoluteDateFilter method), 181	__init__ (gooddata_sdk.type_converter.Converter method), 202
__init__ (gooddata_sdk.compute.model.filter.AllTimeFilter method), 182	__init__ (gooddata_sdk.type_converter.ConverterRegistryStore method), 203
__init__ (gooddata_sdk.compute.model.filter.AttributeFilter method), 183	__init__ (gooddata_sdk.type_converter.DBTypeConverterStore method), 204
__init__ (gooddata_sdk.compute.model.filter.MetricValueFilter method), 183	__init__ (gooddata_sdk.type_converter.DateConverter method), 204
__init__ (gooddata_sdk.compute.model.filter.NegativeAttributeFilter. method), 184	__init__ (gooddata_sdk.type_converter.DatetimeConverter method), 205
__init__ (gooddata_sdk.compute.model.filter.PositiveAttributeFilter. method), 185	__init__ (gooddata_sdk.type_converter.IntegerConverter method), 206
__init__ (gooddata_sdk.compute.model.filter.RankingFilter method), 185	__init__ (gooddata_sdk.type_converter.StringConverter method), 207
__init__ (gooddata_sdk.compute.model.filter.RelativeDateFilter method), 207	__init__ (gooddata_sdk.type_converter.TypeConverterRegistry method), 207

__init__() (gooddata_sdk.utils.AllPagedEntities method), 210
 __init__() (gooddata_sdk.utils.IndentDumper method), 211
 __init__() (gooddata_sdk.utils.SideLoads method), 216

A

AbsoluteDateFilter (class in gooddata_sdk.compute.model.filter), 181
 aggregation(gooddata_sdk.compute.model.execution.TotalDefinition attribute), 180
 AllPagedEntities (class in gooddata_sdk.utils), 210
 AllTimeFilter (class in gooddata_sdk.compute.model.filter), 182
 ArithmeticMetric (class in gooddata_sdk.compute.model.metric), 187
 Attribute (class in gooddata_sdk.compute.model.attribute), 173
 AttributeConverterStore (class in gooddata_sdk.type_converter), 201
 AttributeFilter (class in gooddata_sdk.compute.model.filter), 183

B

BareExecutionResponse (class in gooddata_sdk.compute.model.execution), 176
 Base (class in gooddata_sdk.catalog.base), 33
 BasicCredentials (class in gooddata_sdk.catalog.entity), 74
 BigQueryAttributes (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 56
 build_stores() (in module gooddata_sdk.type_converter), 201

C

camel_to_snake() (in module gooddata_sdk.utils), 209
 catalog_with_valid_objects() (gooddata_sdk.catalog.workspace.model_container.CatalogWorkspaceContent method), 168
 CatalogAnalyticsBase (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model), 117
 CatalogAssigneeIdentifier (class in gooddata_sdk.catalog.identifier), 79
 CatalogAttribute (class in gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset), 156
 CatalogDataset (class in gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset), 157
 CatalogDataSource (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 57
 CatalogDataSourceBigQuery (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 58
 CatalogDataSourcePostgres (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 60
 CatalogDataSourceRedshift (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 62
 CatalogDataSourceService (class in gooddata_sdk.catalog.data_source.service), 71
 CatalogDataSourceSnowflake (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 64
 CatalogDataSourceTable (class in gooddata_sdk.catalog.data_source.entity_model.content_objects.table), 51
 CatalogDataSourceTableAttributes (class in gooddata_sdk.catalog.data_source.entity_model.content_objects.table), 53
 CatalogDataSourceTableColumn (class in gooddata_sdk.catalog.data_source.entity_model.content_objects.table), 54
 CatalogDataSourceTableIdentifier (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model), 130
 CatalogDataSourceVertica (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 66
 CatalogDeclarativeAnalyticalDashboard (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model), 119
 CatalogDeclarativeAnalytics (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model), 120
 CatalogDeclarativeAnalyticsLayer (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model), 122
 CatalogDeclarativeAttribute (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model), 131
 CatalogDeclarativeColumn (class in gooddata_sdk.catalog.data_source.declarative_model.physical_model.physical_model), 45
 CatalogDeclarativeDashboardPlugin (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model), 124
 CatalogDeclarativeDataset (class in gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model), 124

134		<code>data_sdk.catalog.user.declarative_model.user_group)</code> ,
CatalogDeclarativeDataSource	(class in good- <code>data_sdk.catalog.data_source.declarative_model</code>),	101
41		CatalogDeclarativeUsers (class in good- <code>data_sdk.catalog.user.declarative_model.user</code>),
CatalogDeclarativeDataSourcePermission	(class in good- <code>data_sdk.catalog.permission.declarative_model.permission</code>),	97
90		CatalogDeclarativeUsersUserGroups (class in good- <code>data_sdk.catalog.user.declarative_model.user_and_user_groups</code>)
CatalogDeclarativeDataSources	(class in good- <code>data_sdk.catalog.data_source.declarative_model</code>),	98
43		CatalogDeclarativeVisualizationObject (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.analyt</code>
CatalogDeclarativeDateDataset	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset</code>),	138
140		CatalogDeclarativeWorkspace (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeFact	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset</code>),	137
136		CatalogDeclarativeWorkspaceDataFilter (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeFilterContext	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model</code>),	125
125		CatalogDeclarativeWorkspaceDataFilters (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeLabel	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset</code>),	137
137		CatalogDeclarativeWorkspaceDataFilterSetting (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeLdm	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model</code>),	152
144		CatalogDeclarativeWorkspaceDataFilterSetting (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeMetric	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model</code>),	126
126		CatalogDeclarativeWorkspaceHierarchyPermission (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeModel	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm</code>),	145
145		CatalogDeclarativeWorkspaceModel (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeReference	(class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm</code>),	139
139		CatalogDeclarativeWorkspacePermissions (class in good- <code>data_sdk.catalog.permission.declarative_model.permission</code>),
CatalogDeclarativeSetting	(class in good- <code>data_sdk.catalog.setting</code>),	95
95		CatalogDeclarativeWorkspacePermissions (class in good- <code>data_sdk.catalog.permission.declarative_model.permission</code>),
CatalogDeclarativeSingleWorkspacePermission	(class in good- <code>data_sdk.catalog.permission.declarative_model.permission</code>),	93
91		CatalogDeclarativeWorkspaces (class in good- <code>data_sdk.catalog.workspace.declarative_model.workspace.worksp</code>
CatalogDeclarativeTable	(class in good- <code>data_sdk.catalog.data_source.declarative_model</code>),	154
49		CatalogDependentEntitiesGraph (class in good- <code>data_sdk.catalog.workspace.entity_model.graph_objects.graph</code>),
CatalogDeclarativeTables	(class in good- <code>data_sdk.catalog.data_source.declarative_model</code>),	161
47		CatalogDependentEntitiesNode (class in good- <code>data_sdk.catalog.workspace.entity_model.graph_objects.graph</code>),
CatalogDeclarativeUser	(class in good- <code>data_sdk.catalog.user.declarative_model.user</code>),	162
96		CatalogDependentEntitiesRequest (class in good- <code>data_sdk.catalog.workspace.entity_model.graph_objects.graph</code>),
CatalogDeclarativeUserGroup	(class in good- <code>data_sdk.catalog.user.declarative_model.user_group</code>),	163
100		CatalogDependentEntitiesResponse (class in good- <code>data_sdk.catalog.workspace.entity_model.graph_objects.graph</code>),
CatalogDeclarativeUserGroups	(class in good-	164

CatalogEntity (class in gooddata_sdk.catalog.entity), 75	CatalogUser (class in gooddata_sdk.catalog.user.entity_model.user), 102
CatalogEntityIdentifier (class in gooddata_sdk.catalog.workspace.entity_model.graph_catalog_entity), 165	CatalogUserAttributes (class in gooddata_sdk.catalog.user.entity_model.user), 103
CatalogFact (class in gooddata_sdk.catalog.workspace.entity_model.content_catalog_fact), 158	CatalogUserDocument (class in gooddata_sdk.catalog.user.entity_model.user), 104
CatalogGenerateLdmRequest (class in gooddata_sdk.catalog.data_source.action_requests.ldm_catalog_generate), 35	CatalogUserGroup (class in gooddata_sdk.catalog.user.entity_model.user_group), 108
CatalogGrainIdentifier (class in gooddata_sdk.catalog.identifier), 79	CatalogUserGroupDocument (class in gooddata_sdk.catalog.user.entity_model.user_group), 109
CatalogGranularitiesFormatting (class in gooddata_sdk.catalog.workspace.declarative_model.workspace_logical_model.date_dataset.date_dataset), 143	CatalogUserGroupIdentifier (class in gooddata_sdk.catalog.identifier), 82
CatalogLabel (class in gooddata_sdk.catalog.workspace.entity_model.content_catalog_label), 159	CatalogUserGroupParents (class in gooddata_sdk.catalog.user.entity_model.user_group), 110
CatalogLabelIdentifier (class in gooddata_sdk.catalog.identifier), 80	CatalogUserGroupRelationships (class in gooddata_sdk.catalog.user.entity_model.user_group), 106
CatalogMetric (class in gooddata_sdk.catalog.workspace.entity_model.content_objects.metric), 160	CatalogUserGroupsData (class in gooddata_sdk.catalog.user.entity_model.user), 105
CatalogNameEntity (class in gooddata_sdk.catalog.entity), 75	CatalogUserRelationships (class in gooddata_sdk.catalog.user.entity_model.user), 106
CatalogOrganization (class in gooddata_sdk.catalog.organization.entity_model.organization), 84	CatalogUserService (class in gooddata_sdk.catalog.user.service), 112
CatalogOrganizationAttributes (class in gooddata_sdk.catalog.organization.entity_model.organization), 85	CatalogWorkspace (class in gooddata_sdk.catalog.workspace.entity_model.workspace), 166
CatalogOrganizationDocument (class in gooddata_sdk.catalog.organization.entity_model.organization), 87	CatalogWorkspaceContent (class in gooddata_sdk.catalog.workspace.model_container), 167
CatalogOrganizationService (class in gooddata_sdk.catalog.organization.service), 88	CatalogWorkspaceContentService (class in gooddata_sdk.catalog.workspace.content_service), 114
CatalogPermissionService (class in gooddata_sdk.catalog.permission.service), 94	CatalogWorkspaceIdentifier (class in gooddata_sdk.catalog.identifier), 83
CatalogReferenceIdentifier (class in gooddata_sdk.catalog.identifier), 81	CatalogWorkspaceService (class in gooddata_sdk.catalog.workspace.service), 169
CatalogScanModelRequest (class in gooddata_sdk.catalog.data_source.action_requests.scan_model_request), 39	change_case() (in module gooddata_sdk.utils), 209
CatalogScanResultPdm (class in gooddata_sdk.catalog.data_source.declarative_model.physical_model_helper), 48	change_case_helper() (in module gooddata_sdk.utils), 209
CatalogServiceBase (class in gooddata_sdk.catalog.catalog_service_base), 34	column_ids (gooddata_sdk.table.ExecutionTable property), 200
CatalogTitleEntity (class in gooddata_sdk.catalog.entity), 76	column_metadata (gooddata_sdk.table.ExecutionTable property), 200
CatalogTypeEntity (class in gooddata_sdk.catalog.entity), 76	compute_model_to_api_model() (in module gooddata_sdk.compute.model.execution), 175
	compute_valid_objects() (good-

data_sdk.catalog.workspace.content_service.CatalogWorkspaceContentService (good-
method), 116

ComputeService (class in good-
data_sdk.compute.service), 191

Converter (class in *gooddata_sdk.type_converter*), 202

converter() (*gooddata_sdk.type_converter.TypeConverterRegistry* class method), 201
method), 207

ConverterRegistryStore (class in good-
data_sdk.type_converter), 203

count() (*gooddata_sdk.utils.AllPagedEntities* method), 211

create() (*gooddata_sdk.sdk.GoodDataSdk* class
method), 198

create_directory() (in module *gooddata_sdk.utils*),
209

Credentials (class in *gooddata_sdk.catalog.entity*), 76

D

data (*gooddata_sdk.utils.AllPagedEntities* property),
211

DatabaseAttributes (class in good-
data_sdk.catalog.data_source.entity_model.data_source),
68

DataSourceValidator (class in good-
data_sdk.catalog.data_source.validation.data_source),
73

DateConverter (class in *gooddata_sdk.type_converter*),
204

DatetimeConverter (class in good-
data_sdk.type_converter), 205

DBTypeConverterStore (class in good-
data_sdk.type_converter), 204

delete_workspace() (good-
data_sdk.catalog.workspace.service.CatalogWorkspaceService
method), 171

E

ExecModelEntity (class in good-
data_sdk.compute.model.base), 174

Execution (class in good-
data_sdk.compute.model.execution), 177

ExecutionDefinition (class in good-
data_sdk.compute.model.execution), 177

ExecutionResponse (in module good-
data_sdk.compute.model.execution), 178

ExecutionResult (class in good-
data_sdk.compute.model.execution), 178

ExecutionTable (class in *gooddata_sdk.table*), 199

F

Filter (class in *gooddata_sdk.compute.model.base*),
174

find_work_data_center() (good-
data_sdk.catalog.workspace.entity_model.content_objects.database
method), 158

find_converter() (good-
data_sdk.type_converter.AttributeConverterStore
class method), 201

find_converter() (good-
data_sdk.type_converter.ConverterRegistryStore
class method), 203

find_converter() (good-
data_sdk.type_converter.DBTypeConverterStore
class method), 204

find_label_attribute() (good-
data_sdk.catalog.workspace.model_container.CatalogWorkspace
method), 168

for_exec_def() (good-
data_sdk.compute.service.ComputeService
method), 192

from_api() (*gooddata_sdk.catalog.base.Base* class
method), 33

from_api() (*gooddata_sdk.catalog.data_source.action_requests.ldm_request*
class method), 38

from_api() (*gooddata_sdk.catalog.data_source.action_requests.scan_model*
class method), 40

from_api() (*gooddata_sdk.catalog.data_source.declarative_model.data_source*
class method), 43

from_api() (*gooddata_sdk.catalog.data_source.declarative_model.data_source*
class method), 44

from_api() (*gooddata_sdk.catalog.data_source.declarative_model.physical_model*
class method), 46

from_api() (*gooddata_sdk.catalog.data_source.declarative_model.physical_model*
class method), 48

from_api() (*gooddata_sdk.catalog.data_source.declarative_model.physical_model*
class method), 49

from_api() (*gooddata_sdk.catalog.data_source.declarative_model.physical_model*
class method), 50

from_api() (*gooddata_sdk.catalog.data_source.entity_model.content_objects*
class method), 52

from_api() (*gooddata_sdk.catalog.data_source.entity_model.content_objects*
class method), 53

from_api() (*gooddata_sdk.catalog.data_source.entity_model.content_objects*
class method), 55

from_api() (*gooddata_sdk.catalog.identifier.CatalogAssigneeIdentifier*
class method), 79

from_api() (*gooddata_sdk.catalog.identifier.CatalogGrainIdentifier*
class method), 80

from_api() (*gooddata_sdk.catalog.identifier.CatalogLabelIdentifier*
class method), 81

from_api() (*gooddata_sdk.catalog.identifier.CatalogReferenceIdentifier*
class method), 82

from_api() (*gooddata_sdk.catalog.identifier.CatalogUserGroupIdentifier*
class method), 83

from_api() (*gooddata_sdk.catalog.identifier.CatalogWorkspaceIdentifier*
class method), 83

`from_dict()` (`gooddata_sdk.catalog.workspace.declarative_model.ldm.CatalogDeclarativeModel`
class method), 146
`from_dict()` (`gooddata_sdk.catalog.workspace.declarative_model.phbase.CatalogDeclarativeWorkspace`
class method), 148
`from_dict()` (`gooddata_sdk.catalog.workspace.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilter`
class method), 150
`from_dict()` (`gooddata_sdk.catalog.workspace.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilters`
class method), 153
`from_dict()` (`gooddata_sdk.catalog.workspace.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilterSet`
class method), 152
`from_dict()` (`gooddata_sdk.catalog.workspace.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilterSetAddn_request`
class method), 154
`from_dict()` (`gooddata_sdk.catalog.workspace.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilterSetAddn_request.scan_mode`
class method), 155
`from_dict()` (`gooddata_sdk.catalog.workspace.entity_model.CatalogDataSourceEntityModel`
class method), 162
`from_dict()` (`gooddata_sdk.catalog.workspace.entity_model.CatalogDataSourceEntityModel.data_source`
class method), 163
`from_dict()` (`gooddata_sdk.catalog.workspace.entity_model.CatalogDataSourceEntityModel.physical`
class method), 164
`from_dict()` (`gooddata_sdk.catalog.workspace.entity_model.CatalogDataSourceEntityModel.physical`
class method), 165
`from_dict()` (`gooddata_sdk.catalog.workspace.entity_model.CatalogEntityIdentifier`
class method), 166

G

`get_dataset()` (`gooddata_sdk.catalog.data_source.declarative_model.phbase.CatalogDeclarativeModel.physical`
method), 168
`get_exec_metadata()` (`gooddata_sdk.catalog.data_source.entity_model.CatalogWorkspaceContent`
method), 192
`get_full_catalog()` (`gooddata_sdk.catalog.data_source.entity_model.CatalogWorkspaceContent`
method), 116
`get_insight()` (`gooddata_sdk.insight.InsightService`
method), 196
`get_insights()` (`gooddata_sdk.insight.InsightService`
method), 196
`get_metric()` (`gooddata_sdk.catalog.workspace.model_container.CatalogWorkspaceContent`
method), 168
`get_pdm_folder()` (in module `gooddata_sdk.catalog.data_source.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilter`
47), 47
`get_sorted_yaml_files()` (in module `gooddata_sdk.catalog.data_source.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilter`
209), 209
`get_workspace()` (`gooddata_sdk.catalog.data_source.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilter`
method), 171
`get_workspace_folder()` (in module `gooddata_sdk.catalog.data_source.declarative_model.phbase.CatalogDeviceHubWorkspaceDataFilter`
146), 146
`gooddata_sdk`
module, 31

gooddata_sdk.catalog.permission.declarative_model module, 89
 gooddata_sdk.catalog.permission.declarative_model.permission module, 89
 gooddata_sdk.catalog.permission.service module, 94
 gooddata_sdk.catalog.setting module, 95
 gooddata_sdk.catalog.types module, 96
 gooddata_sdk.catalog.user module, 96
 gooddata_sdk.catalog.user.declarative_model module, 96
 gooddata_sdk.catalog.user.declarative_model.user module, 96
 gooddata_sdk.catalog.user.declarative_model.user_group module, 98
 gooddata_sdk.catalog.user.declarative_model.user_group_group module, 99
 gooddata_sdk.catalog.user.entity_model module, 102
 gooddata_sdk.catalog.user.entity_model.user module, 102
 gooddata_sdk.catalog.user.entity_model.user_group module, 107
 gooddata_sdk.catalog.user.service module, 111
 gooddata_sdk.catalog.workspace module, 114
 gooddata_sdk.catalog.workspace.content_service module, 114
 gooddata_sdk.catalog.workspace.declarative_model module, 116
 gooddata_sdk.catalog.workspace.declarative_model.declarative_model module, 116
 gooddata_sdk.catalog.workspace.declarative_model.insights module, 117
 gooddata_sdk.catalog.workspace.declarative_model.insights.analytics_model module, 117
 gooddata_sdk.catalog.workspace.declarative_model.logical_model module, 129
 gooddata_sdk.catalog.workspace.declarative_model.logical_model.dataset module, 130
 gooddata_sdk.catalog.workspace.declarative_model.logical_model.type_converter module, 130
 gooddata_sdk.catalog.workspace.declarative_model.logical_model.util module, 140
 gooddata_sdk.catalog.workspace.declarative_model.logical_model.util.date_dataset module, 140
 gooddata_sdk.catalog.workspace.declarative_model.logical_model.util.date_dataset.date_dataset module, 140
 gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm module, 144
 gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace module, 146
 gooddata_sdk.catalog.workspace.entity_model module, 155
 gooddata_sdk.catalog.workspace.entity_model.content_object module, 156
 gooddata_sdk.catalog.workspace.entity_model.content_object.content_object module, 156
 gooddata_sdk.catalog.workspace.entity_model.content_object.content_object_content module, 160
 gooddata_sdk.catalog.workspace.entity_model.graph_objects module, 161
 gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph_objects module, 161
 gooddata_sdk.catalog.workspace.entity_model.workspace module, 166
 gooddata_sdk.catalog.workspace.model_container module, 167
 gooddata_sdk.catalog.workspace.service module, 169
 gooddata_sdk.client module, 171
 gooddata_sdk.compute module, 172
 gooddata_sdk.compute.model module, 172
 gooddata_sdk.compute.model.attribute module, 173
 gooddata_sdk.compute.model.base module, 173
 gooddata_sdk.compute.model.execution module, 175
 gooddata_sdk.compute.model.filter module, 181
 gooddata_sdk.compute.model.metric module, 187
 gooddata_sdk.compute.service module, 191
 gooddata_sdk.insights module, 192
 gooddata_sdk.insights.analytics_model module, 197
 gooddata_sdk.logical_model module, 198
 gooddata_sdk.logical_model.dataset module, 199
 gooddata_sdk.logical_model.type_converter module, 201
 gooddata_sdk.logical_model.util module, 208
 gooddata_sdk.logical_model.util.date_dataset module, 208
 GoodDataSdk (class in gooddata_sdk.sdk), 197
 to_obj_to_key() (in module gooddata_sdk.utils), 209

idx (*gooddata_sdk.compute.model.execution.TotalDimension attribute*), 180

included (*gooddata_sdk.utils.AllPagedEntities property*), 211

IndentDumper (*class in gooddata_sdk.utils*), 211

index() (*gooddata_sdk.utils.AllPagedEntities method*), 211

Insight (*class in gooddata_sdk.insight*), 193

InsightAttribute (*class in gooddata_sdk.insight*), 194

InsightBucket (*class in gooddata_sdk.insight*), 194

InsightFilter (*class in gooddata_sdk.insight*), 195

InsightMetric (*class in gooddata_sdk.insight*), 195

InsightService (*class in gooddata_sdk.insight*), 196

IntegerConverter (*class in gooddata_sdk.type_converter*), 206

is_available (*gooddata_sdk.support.SupportService property*), 198

items (*gooddata_sdk.compute.model.execution.TotalDimension attribute*), 180

L

load_all_entities() (*in module gooddata_sdk.utils*), 209

load_all_entities_dict() (*in module gooddata_sdk.utils*), 210

local_id (*gooddata_sdk.compute.model.execution.TotalDefinition attribute*), 180

M

Metric (*class in gooddata_sdk.compute.model.metric*), 188

metric_local_id (*gooddata_sdk.compute.model.execution.TotalDefinition attribute*), 180

MetricValueFilter (*class in gooddata_sdk.compute.model.filter*), 183

module

- gooddata_sdk, 31
- gooddata_sdk.catalog, 32
- gooddata_sdk.catalog.base, 32
- gooddata_sdk.catalog.catalog_service_base, 34
- gooddata_sdk.catalog.data_source, 34
- gooddata_sdk.catalog.data_source.action_requests, 35
- gooddata_sdk.catalog.data_source.action_requests.ldm_request, 35
- gooddata_sdk.catalog.data_source.action_requests.scan_model_request, 38
- gooddata_sdk.catalog.data_source.declarative_model, 40
- gooddata_sdk.catalog.data_source.declarative_model.data_source, 41
- gooddata_sdk.catalog.data_source.declarative_model.physical_model, 44
- gooddata_sdk.catalog.data_source.declarative_model.physical_model.content_model, 45
- gooddata_sdk.catalog.data_source.declarative_model.physical_model.entity_model, 46
- gooddata_sdk.catalog.data_source.declarative_model.physical_model.entity_model.content_model, 49
- gooddata_sdk.catalog.data_source.declarative_model.physical_model.entity_model.content_model.data_source, 51
- gooddata_sdk.catalog.data_source.declarative_model.physical_model.entity_model.content_model.data_source.service, 51
- gooddata_sdk.catalog.data_source.declarative_model.physical_model.entity_model.data_source, 56
- gooddata_sdk.catalog.data_source.service, 70
- gooddata_sdk.catalog.data_source.validation, 73
- gooddata_sdk.catalog.data_source.validation.data_source, 73
- gooddata_sdk.catalog.entity, 74
- gooddata_sdk.catalog.identifier, 78
- gooddata_sdk.catalog.organization, 84
- gooddata_sdk.catalog.organization.entity_model, 84
- gooddata_sdk.catalog.organization.entity_model.organization, 84
- gooddata_sdk.catalog.organization.service, 88
- gooddata_sdk.catalog.permission, 89
- gooddata_sdk.catalog.permission.declarative_model, 89
- gooddata_sdk.catalog.permission.declarative_model.permission, 89
- gooddata_sdk.catalog.permission.service, 94
- gooddata_sdk.catalog.setting, 95
- gooddata_sdk.catalog.types, 96
- gooddata_sdk.catalog.user, 96
- gooddata_sdk.catalog.user.declarative_model, 96
- gooddata_sdk.catalog.user.declarative_model.user, 96
- gooddata_sdk.catalog.user.declarative_model.user_and_group, 98
- gooddata_sdk.catalog.user.declarative_model.user_group, 99
- gooddata_sdk.catalog.user.entity_model, 102
- gooddata_sdk.catalog.user.entity_model.user, 102
- gooddata_sdk.catalog.user.entity_model.user_group, 102

107
 gooddata_sdk.catalog.user.service, 111
 gooddata_sdk.catalog.workspace, 114
 gooddata_sdk.catalog.workspace.content_service, 114
 gooddata_sdk.catalog.workspace.declarative_model, 116
 gooddata_sdk.catalog.workspace.declarative_model.workspace, 116
 gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model, 117
 gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model, 117
 gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model, 129
 gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset, 130
 gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset, 140
 gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset, 140
 gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.idm, 144
 gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.idm.dataset, 146
 gooddata_sdk.catalog.workspace.entity_model, 155
 gooddata_sdk.catalog.workspace.entity_model.content_objects, 156
 gooddata_sdk.catalog.workspace.entity_model.content_objects.dataset, 156
 gooddata_sdk.catalog.workspace.entity_model.content_objects.metric, 160
 gooddata_sdk.catalog.workspace.entity_model.graph_objects, 161
 gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph, 161
 gooddata_sdk.catalog.workspace.entity_model.workspace, 166
 gooddata_sdk.catalog.workspace.model_container, 167
 gooddata_sdk.catalog.workspace.service, 169
 gooddata_sdk.client, 171
 gooddata_sdk.compute, 172
 gooddata_sdk.compute.model, 172
 gooddata_sdk.compute.model.attribute, 173
 gooddata_sdk.compute.model.base, 173
 gooddata_sdk.compute.model.execution, 175
 gooddata_sdk.compute.model.filter, 181
 gooddata_sdk.compute.model.metric, 187
 gooddata_sdk.compute.service, 191
 gooddata_sdk.insight, 192
 gooddata_sdk.sdk, 197
 gooddata_sdk.support, 198
 gooddata_sdk.table, 199
 gooddata_sdk.type_converter, 201
 gooddata_sdk.utils, 208

N
 NegativeAttributeFilter (class in gooddata_sdk.compute.model.filter), 184

O
 ObjId (class in gooddata_sdk.compute.model.base), 174
 one_scan_true() (in module gooddata_sdk.catalog.data_source.action_requests.scan_model_request), 39

P
 PopDate (class in gooddata_sdk.compute.model.metric), 188
 PopDateDataset (class in gooddata_sdk.compute.model.metric), 189
 PopDateMetric (class in gooddata_sdk.compute.model.metric), 189
 PopDateSetMetric (class in gooddata_sdk.compute.model.metric), 190
 PositiveAttributeFilter (class in gooddata_sdk.compute.model.filter), 185
 PostgresAttributes (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 68

R
 RankingFilter (class in gooddata_sdk.compute.model.filter), 185
 read_all() (gooddata_sdk.table.ExecutionTable method), 200
 read_layout_from_file() (in module gooddata_sdk.utils), 210
 read_result() (gooddata_sdk.compute.model.execution.BareExecutionResponse method), 176
 RedshiftAttributes (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 69
 register() (gooddata_sdk.type_converter.AttributeConverterStore class method), 202
 register() (gooddata_sdk.type_converter.ConverterRegistryStore class method), 203
 register() (gooddata_sdk.type_converter.DBTypeConverterStore class method), 204
 register() (gooddata_sdk.type_converter.TypeConverterRegistry method), 208

RelativeDateFilter (class in gooddata_sdk.compute.model.filter), 186

reset() (gooddata_sdk.type_converter.AttributeConverter class method), 202

reset() (gooddata_sdk.type_converter.ConverterRegistry class method), 203

reset() (gooddata_sdk.type_converter.DBTypeConverter class method), 204

ResultSizeLimitsExceeded, 181

S

SideLoads (class in gooddata_sdk.utils), 216

SimpleMetric (class in gooddata_sdk.compute.model.metric), 191

snake_to_camel() (in module gooddata_sdk.utils), 210

SnowflakeAttributes (class in gooddata_sdk.catalog.data_source.entity_model.data_source), 69

StringConverter (class in gooddata_sdk.type_converter), 207

SupportService (class in gooddata_sdk.support), 198

T

TableService (class in gooddata_sdk.table), 200

time_comparison_master (gooddata_sdk.insight.InsightMetric property), 196

to_date() (gooddata_sdk.type_converter.DateConverter class method), 205

to_datetime() (gooddata_sdk.type_converter.DatetimeConverter class method), 206

to_dict() (gooddata_sdk.catalog.base.Base method), 33

to_dict() (gooddata_sdk.catalog.data_source.action_requests.tam_request.CatalogGenerateTamRequest method), 38

to_dict() (gooddata_sdk.catalog.data_source.action_requests.scan_model_request.CatalogScanModelRequest method), 40

to_dict() (gooddata_sdk.catalog.data_source.declarative_model.data_source.CatalogDeclarativeDataSource method), 43

to_dict() (gooddata_sdk.catalog.data_source.declarative_model.data_source.CatalogDeclarativeDataSources method), 44

to_dict() (gooddata_sdk.catalog.data_source.declarative_model.physical_model.column.CatalogDeclarativeColumn method), 46

to_dict() (gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.CatalogDeclarativeTable method), 48

to_dict() (gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.CatalogScanResultTam method), 49

to_dict() (gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.CatalogDeclarativeTable method), 51

to_dict() (gooddata_sdk.catalog.data_source.entity_model.comment_objects.table.CatalogDataSourceTable method), 52

to_dict() (gooddata_sdk.catalog.data_source.entity_model.comment_objects.table.CatalogDataSourceTableAttributes method), 54

to_dict() (gooddata_sdk.catalog.data_source.entity_model.content_objects.table.CatalogDataSourceTable method), 55

to_dict() (gooddata_sdk.catalog.identifier.CatalogAssigneeIdentifier method), 79

to_dict() (gooddata_sdk.catalog.identifier.CatalogGrainIdentifier method), 80

to_dict() (gooddata_sdk.catalog.identifier.CatalogLabelIdentifier method), 81

to_dict() (gooddata_sdk.catalog.identifier.CatalogReferenceIdentifier method), 82

to_dict() (gooddata_sdk.catalog.identifier.CatalogUserGroupIdentifier method), 83

to_dict() (gooddata_sdk.catalog.identifier.CatalogWorkspaceIdentifier method), 83

to_dict() (gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganization method), 85

to_dict() (gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganizationAttributes method), 87

to_dict() (gooddata_sdk.catalog.organization.entity_model.organization.CatalogOrganizationTable method), 88

to_dict() (gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativePermission method), 90

to_dict() (gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativePermissions method), 91

to_dict() (gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativePermissionsTable method), 92

to_dict() (gooddata_sdk.catalog.permission.declarative_model.permission.CatalogDeclarativePermissionsTableAttributes method), 93

to_dict() (gooddata_sdk.catalog.setting.CatalogDeclarativeSetting method), 95

to_dict() (gooddata_sdk.catalog.user.declarative_model.user.CatalogDeclarativeUser method), 97

to_dict() (gooddata_sdk.catalog.user.declarative_model.user.CatalogDeclarativeUsers method), 98

to_dict() (gooddata_sdk.catalog.user.declarative_model.user_and_user_group.CatalogDeclarativeUserAndUserGroup method), 99

to_dict() (gooddata_sdk.catalog.user.declarative_model.user_group.CatalogDeclarativeUserGroup method), 100

to_dict() (gooddata_sdk.catalog.user.declarative_model.user_group.CatalogDeclarativeUserGroups method), 101

to_dict() (gooddata_sdk.catalog.user.entity_model.user.CatalogUser method), 103

to_dict() (gooddata_sdk.catalog.user.entity_model.user.CatalogUserAttributes method), 104

to_dict() (gooddata_sdk.catalog.user.entity_model.user.CatalogUserDoc method), 105

to_dict() (gooddata_sdk.catalog.user.entity_model.user.CatalogUserGroup method), 106

to_dict() (gooddata_sdk.catalog.user.entity_model.user.CatalogUserRelationship method), 107

to_dict() (gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroup method), 108

to_dict() (gooddata_sdk.catalog.user.entity_model.user_group.CatalogUserGroupAttributes method), 109

- to_dict() (gooddata_sdk.catalog.user.entity_model.user_group_model.GlobalGroupRoleCatalog.workspace.entity_model.graph_objects.
method), 110
- to_dict() (gooddata_sdk.catalog.user.entity_model.user_group_model.GlobalGroupRelationshipCatalog.workspace.entity_model.graph_objects.
method), 111
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.GlobalGroupRoleCatalog.workspace.declarative_model.GlobalGroupRoleCatalog.
method), 118
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.GlobalGroupRelationshipCatalog.workspace.declarative_model.GlobalGroupRelationshipCatalog.
method), 120
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.GlobalGroupRoleCatalog.workspace.declarative_model.GlobalGroupRoleCatalog.
method), 121
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.GlobalGroupRelationshipCatalog.workspace.declarative_model.GlobalGroupRelationshipCatalog.
method), 123
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.GlobalGroupRoleCatalog.workspace.declarative_model.GlobalGroupRoleCatalog.
method), 125
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.GlobalGroupRelationshipCatalog.workspace.declarative_model.GlobalGroupRelationshipCatalog.
method), 126
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.GlobalGroupRoleCatalog.workspace.declarative_model.GlobalGroupRoleCatalog.
method), 128
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarativeVisualizations.
method), 129
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDataSourceTableIdentifier.
method), 131
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarativeAttribute.
method), 133
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarativeDataset.
method), 135
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarativeFact.
method), 137
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarativeLabel.
method), 138
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.dataset.dataset.CatalogDeclarativeReference.
method), 140
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset.CatalogDeclarativeDateDataset.
method), 142
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.date_dataset.date_dataset.CatalogGranularity.
method), 143
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm.CatalogDeclarativeLdm.
method), 145
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.logical_model.ldm.CatalogDeclarativeModel.
method), 146
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspace.
method), 148
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilter.
method), 150
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilters.
method), 153
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceDataFilterSettings.
method), 152
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaceModel.
method), 154
- to_dict() (gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.CatalogDeclarativeWorkspaces.
method), 155
- to_dict() (gooddata_sdk.catalog.workspace.entity_model.graph_objects.graph.CatalogDependentEntitiesGraph.
method), 162