GoodData SDK

Release 1.2.0

GoodData Corporation

Mar 01, 2023

CONTENTS:

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.

CHAPTER

ONE

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**.

CHAPTER

TWO

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.

```
gooddata_layouts
└── organization_id
        ├── workspaces
        │       ├── workspace_a
        │       │       ├── analytics_model
        │       │       │   ├── analytical_dashboards
        │       │       │   │       └── analytical_dashboard.yaml
        │       │       │   ├── dashboard_plugins
        │       │       │   │       └── dashboard_plugin.yaml
        │       │       │   ├── filter_contexts
        │       │       │   │       └── filter_context.yaml
        │       │       │   ├── metrics
        │       │       │   │       └── metric.yaml
        │       │       │   └── visualization_objects
        │       │       │           └── visualization_object.yaml
        │       │       ├── ldm
        │       │       │   ├── datasets
        │       │       │   │       └── dataset.yaml
        │       │       │   └── date_instances
        │       │       │           └── date_instance.yaml
        │       │       └── workspace_a.yaml
        │       └── workspace_b
        │               └── ...
        │
        └── workspaces_data_filters
```

(continues on next page)

(continued from previous page)

```
                ├── filter_1.yaml
                └── filter_2.yaml
```

- `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.

```
gooddata_layouts
└── organization_id
        └── workspaces
                └── workspace_a
                        ├── analytics_model
                        │   ├── analytical_dashboards
                        │   │       └── analytical_dashboard.yaml
                        │   ├── dashboard_plugins
                        │   │       └── dashboard_plugin.yaml
                        │   ├── filter_contexts
                        │   │       └── filter_context.yaml
                        │   ├── metrics
                        │   │       └── metric.yaml
                        │   └── visualization_objects
                        │           └── visualization_object.yaml
                        └── ldm
                            ├── datasets
                            │       └── dataset.yaml
                            └── date_instances
                                    └── date_instance.yaml
```

- `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

```python
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)

(continued from previous 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_ldm_to_disk(self, workspace_id: str, path: Path = Path.cwd())`

  Store the ldm layout in the directory for a given workspace. The directory structure below shows the output for the path set to `Path("ldm_location")`.

```
ldm_location
     └── ldm
          ├── datasets
          │        └── dataset.yaml
          └── date_instances
                   └── date_instance.yaml
```

- `load_ldm_from_disk(self, path: Path = Path.cwd())`

  The method is used to load ldm stored to disk using method *store_ldm_to_disk*.

- `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_analytics_model_to_disk(self, workspace_id: str, path: Path = Path.cwd())`

  Store the analytics model layout in the directory for a given workspace. The directory structure below shows the output for the path set to `Path("analytics_model_location")`.

```
analytics_model_location
        └── analytics_model
                ├── analytical_dashboards
                │       └── analytical_dashboard.yaml
                ├── dashboard_plugins
                │       └── dashboard_plugin.yaml
                ├── filter_contexts
                │       └── filter_context.yaml
                ├── metrics
                │       └── metric.yaml
                └── visualization_objects
                        └── visualization_object.yaml
```

- `load_analytics_model_from_disk(self, path: Path = Path.cwd())`

  The method is used to load analytics model stored to disk using method *store_analytics_model_to_disk*.

- `store_declarative_analytics_model(workspace_id: str, layout_root_path: Path = Path.cwd())`

  Store declarative analytics model layout in directory hierarchy.

```
gooddata_layouts
└── organization_id
        └── workspaces
                └── workspace_id
                        └── analytics_model
                                ├── analytical_dashboards
                                │       └── analytical_dashboard.yaml
                                ├── dashboard_plugins
                                │       └── dashboard_plugin.yaml
                                ├── filter_contexts
                                │       └── filter_context.yaml
                                ├── metrics
```

(continues on next page)

(continued from previous page)

```
                                    │       └── metric.yaml
                                    └── visualization_objects
                                            └── visualization_object.yaml
```

- `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"

# 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,
```

(continues on next page)

(continued from previous page)

```
    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",
        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
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(
```

(continues on next page)

(continued from previous page)

```
    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")]
    )
)

# 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_pdm_to_disk(self, datasource_id: str, path: Path = Path.cwd())`

  Store the physical model layout in the directory for a given data source. The directory structure below shows the output for the path set to `Path("pdm_location")`.

```
pdm_location
    └── pdm
        ├── table_A.yaml
        └── table_B.yaml
```

- `load_pdm_from_disk(self, path: Path = Path.cwd())`

  The method is used to load pdm stored to disk using method *store_pdm_to_disk*.

- `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
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,
```

(continues on next page)

(continued from previous page)

```
                                                    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)

(continued from previous 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 from
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.

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

- `for_exec_def(workspace_id: str, exec_def: ExecutionDefinition)`

  Returns *Execution*.

  Starts computation in GoodData.CN workspace, using the provided execution definition.

  Example:

```
from gooddata_sdk import GoodDataSdk, ExecutionDefinition, Attribute, SimpleMetric, ObjId

sdk = GoodDataSdk.create(host, token)
workspace_id = "demo"

exec_def = ExecutionDefinition(
    attributes=[
        Attribute(local_id="region", label="region"),
        Attribute(local_id="product_category", label="products.category"),
        Attribute(local_id="state", label="state"),
    ],
    metrics=[
        SimpleMetric(local_id="price", item=ObjId(id="price", type="fact")),
        SimpleMetric(local_id="order_amount", item=ObjId(id="order_amount", type="metric
↪")),
    ],
    filters=[],
    dimensions=[["region", "state"], ["product_category", "measureGroup"]],
)
execution = sdk.compute.for_exec_def(workspace_id, exec_def)
```

(continues on next page)

(continued from previous page)

```
# currently there is no dedicated service for exporting *Execution* results into XLSX/
↪CSV, however it's possible to run it this way:

from gooddata_api_client.model.tabular_export_request import TabularExportRequest

filename = "export.xlsx"
export_request = sdk.client.actions_api.create_tabular_export(
    workspace_id,
    TabularExportRequest(
        execution_result=execution.result_id,
        file_name=filename,
        format="XLSX",
    )
)

while response := sdk.client.actions_api.get_tabular_export(workspace_id, export_request.
↪export_result, _preload_content=False):
    if (response.status == 202):
        time.sleep(2)
        continue
    elif (response.status == 200):
        with open(filename, 'wb') as out_file:
            out_file.write(response.data)
            break
```

- `retrieve_result_cache_metadata(workspace_id: str, result_id: str)`

  Returns *ResultCacheMetadata*.

  Gets execution result's metadata from GoodData.CN workspace for given execution result ID.

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.

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:

```python
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}
```

CHAPTER

THREE

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 metadata 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

Functions

gooddata_sdk.catalog.base.value_in_allowed

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

__init__()	Method generated by attrs for class Base.
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.
to_api()	
to_dict([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_model
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_model

Modules

gooddata_sdk.catalog.data_source.action_model.requests
gooddata_sdk.catalog.data_source.action_model.responses
gooddata_sdk.catalog.data_source.action_model.sql_column

gooddata_sdk.catalog.data_source.action_model.requests

Modules

gooddata_sdk.catalog.data_source.action_model.requests.ldm_request
gooddata_sdk.catalog.data_source.action_model.requests.scan_model_request
gooddata_sdk.catalog.data_source.action_model.requests.scan_sql_request

gooddata_sdk.catalog.data_source.action_model.requests.ldm_request

Classes

CatalogGenerateLdmRequest(*[, separator, ...])
CatalogPdmLdmRequest(*, sqls)
CatalogPdmSql(*, statement, title, columns)

gooddata_sdk.catalog.data_source.action_model.requests.ldm_request.CatalogGenerateLdmRequest

class gooddata_sdk.catalog.data_source.action_model.requests.ldm_request.**CatalogGenerateLdmRequest**(*, separator: str = '__', generate_l Optional = None, table_p Optional = None, view_ Optional = None, primary_ Optional = None, secondar Optional = None, fact_p Optional = None, date_ Optional = None, grain_ Optional = None, reference_ Optional = 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*, *pdm: Optional[CatalogPdmLdmRequest] = None*) → None

Method generated by attrs for class CatalogGenerateLdmRequest.

Methods

__init__(*[, separator, generate_long_ids, ...])	Method generated by attrs for class CatalogGenerateLdmRequest.
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.
to_api()	
to_dict([camel_case])	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
pdm

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_model.requests.ldm_request.CatalogPdmLdmRequest

class gooddata_sdk.catalog.data_source.action_model.requests.ldm_request.**CatalogPdmLdmRequest**(*, *sqls: List[*Catalog

Bases: *Base*

__init__(*, *sqls: List[*CatalogPdmSql*]*) → None

Method generated by attrs for class CatalogPdmLdmRequest.

Methods

__init__(*, sqls)	Method generated by attrs for class CatalogPdmLdmRequest.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

sqls

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_model.requests.ldm_request.CatalogPdmSql

class gooddata_sdk.catalog.data_source.action_model.requests.ldm_request.**CatalogPdmSql**(*, *statement: str*, *title: str*, *columns: List[SqlColumn]*)

Bases: *Base*

__init__(*, *statement: str*, *title: str*, *columns: List[SqlColumn]*) → None

Method generated by attrs for class CatalogPdmSql.

Methods

__init__(*, statement, title, columns)	Method generated by attrs for class CatalogPdmSql.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

statement
title
columns

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_model.requests.scan_model_request

Functions

one_scan_true(instance, *args)

gooddata_sdk.catalog.data_source.action_model.requests.scan_model_request.one_scan_true

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

Classes

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

gooddata_sdk.catalog.data_source.action_model.requests.scan_model_request.CatalogScanModelRequest

class gooddata_sdk.catalog.data_source.action_model.requests.scan_model_request.**CatalogScanModelRequest**

Bases: *Base*

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

Method generated by attrs for class CatalogScanModelRequest.

Methods

__init__(*[, separator, scan_tables, ...])	Method generated by attrs for class CatalogScanModelRequest.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

separator
scan_tables
scan_views
table_prefix
view_prefix
schemata

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_model.requests.scan_sql_request

Classes

ScanSqlRequest(*, sql)	

gooddata_sdk.catalog.data_source.action_model.requests.scan_sql_request.ScanSqlRequest

class gooddata_sdk.catalog.data_source.action_model.requests.scan_sql_request.**ScanSqlRequest**(*, *sql: str*)

Bases: *Base*

__init__(*, *sql: str*) → None

Method generated by attrs for class ScanSqlRequest.

Methods

__init__(*, sql)	Method generated by attrs for class ScanSqlRequest.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

sql

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_model.responses

Modules

gooddata_sdk.catalog.data_source.action_model.responses.scan_sql_response

gooddata_sdk.catalog.data_source.action_model.responses.scan_sql_response

Classes

ScanSqlResponse(*, columns, data_preview)

gooddata_sdk.catalog.data_source.action_model.responses.scan_sql_response.ScanSqlResponse

class gooddata_sdk.catalog.data_source.action_model.responses.scan_sql_response.**ScanSqlResponse**(*, *columns: List[SqlC* *data_prev* *List[List[*

Bases: *Base*

__init__(*, *columns: List[SqlColumn]*, *data_preview: List[List[str]]*) → None

Method generated by attrs for class ScanSqlResponse.

Methods

__init__(*, columns, data_preview)	Method generated by attrs for class ScanSqlResponse.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

columns
data_preview

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_model.sql_column

Classes

SqlColumn(*, data_type, name)

gooddata_sdk.catalog.data_source.action_model.sql_column.SqlColumn

class gooddata_sdk.catalog.data_source.action_model.sql_column.**SqlColumn**(*, *data_type: str*, *name: str*)

Bases: *Base*

__init__(*, *data_type: str*, *name: str*) → None

Method generated by attrs for class SqlColumn.

Methods

__init__(*, data_type, name)	Method generated by attrs for class SqlColumn.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

data_type
name

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

gooddata_sdk.catalog.data_source.declarative_model.data_source
gooddata_sdk.catalog.data_source.declarative_model.physical_model

gooddata_sdk.catalog.data_source.declarative_model.data_source

Classes

CatalogDeclarativeDataSource(*, id, name, 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*, *name: str*, *type: str*, *url: Optional[* = *None*, *schema str*, *enable_c Optional[* = *None*, *pdm:* Cat-a-logDec a-tiveTa-bles = *Cat-a-logDec a-tiveTa-bles(ta cache_ Optional[* = *None*, *user-name: Optional[* = *None*, *pa-ram-e-ters: Optional[* = *None*, *de-coded_ Optional[* = *None*, *per-*

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeDataSource.

Methods

__init__(*, id, name, type[, url, ...])	Method generated by attrs for class CatalogDeclarativeDataSource.
client_class()	
data_source_folder(data_sources_folder, ...)	
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(data_sources_folder, ...)	
store_to_disk(data_sources_folder)	
to_api([password, token, ...])	
to_dict([camel_case])	Converts object into dictionary.
to_test_request([password, token])	

Attributes

id
name
type
url
schema
enable_caching
pdm
cache_path
username
parameters
decoded_parameters
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_*
List[C

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeDataSources.

Methods

__init__(*, data_sources)	Method generated by attrs for class CatalogDeclarativeDataSources.
client_class()	
data_sources_folder(layout_organization_folder)	
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([credentials])	
to_dict([camel_case])	Converts object into dictionary.

Attributes

data_sources

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

gooddata_sdk.catalog.data_source.declarative_model.physical_model.column
gooddata_sdk.catalog.data_source.declarative_model.physical_model.pdm
gooddata_sdk.catalog.data_source.declarative_model.physical_model.table

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

__init__(*, name, data_type[, ...])	Method generated by attrs for class CatalogDeclarativeColumn.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

name
data_type
is_primary_key
referenced_table_id
referenced_table_column

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

get_pdm_folder(data_source_folder)

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**(*, *table Lis = NO IN*

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeTables.

Methods

__init__(*[, tables])	Method generated by attrs for class CatalogDeclarativeTables.
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(data_source_folder)	
store_to_disk(data_source_folder)	
to_api()	
to_dict([camel_case])	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: CatalogDeclarativeTables = CatalogDeclarativeTables(tables=[])*, *warnings: List[Dict] = NOTHING*)

Bases: *Base*

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

Method generated by attrs for class CatalogScanResultPdm.

Methods

__init__(*[, pdm, warnings])	Method generated by attrs for class CatalogScanResultPdm.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

pdm
warnings

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

CatalogDeclarativeTable(*, id, type, path, ...)

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

class gooddata_sdk.catalog.data_source.declarative_model.physical_model.table.**CatalogDeclarativeTable**(*, id, st, ty, st, pa, La, co, La, na, Op, ti, = N

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

__init__(*, id, type, path, columns[, ...])	Method generated by attrs for class CatalogDeclarativeTable.
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(table_file_path)	
store_to_disk(pdm_folder)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
type
path
columns
name_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.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*, *attributes:* CatalogDataSourceTableAttributes)

Bases: *Base*

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

Method generated by attrs for class CatalogDataSourceTable.

Methods

__init__(*, id, type, attributes)	Method generated by attrs for class CatalogDataSourceTable.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
type
attributes

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

__init__(*, columns[, name_prefix, path, type])	Method generated by attrs for class CatalogDataSourceTableAttributes.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

columns
name_prefix
path
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.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

__init__(*, name, data_type[, ...])	Method generated by attrs for class CatalogDataSourceTableColumn.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

name
data_type
is_primary_key
referenced_table_column
referenced_table_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.data_source.entity_model.data_source

Functions

db_attrs_with_template(instance, *args)

gooddata_sdk.catalog.data_source.entity_model.data_source.db_attrs_with_template

gooddata_sdk.catalog.data_source.entity_model.data_source.**db_attrs_with_template**(*instance:* Catalog-Data-Source, **args: Any*) → None

Classes

CatalogDataSource(*, id, name, type, schema)
CatalogDataSourceBase(*, id, name, type, schema)
CatalogDataSourceBigQuery(*, id, name, schema)
CatalogDataSourceGreenplum(*, id, name, schema)
CatalogDataSourcePostgres(*, id, name, schema)
CatalogDataSourceRedshift(*, id, name, schema)
CatalogDataSourceSnowflake(*, id, name, schema)
CatalogDataSourceVertica(*, id, name, schema)
DatabaseAttributes()
GreenplumAttributes(*, host, db_name[, port])
PostgresAttributes(*, host, db_name[, port])
RedshiftAttributes(*, host, db_name[, port])
SnowflakeAttributes(*, account, warehouse, ...)
VerticaAttributes(*, host, db_name[, port])

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*, *type: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[str, str]]] = None*, *decoded_parameters: Optional[List[Dict[str, str]]] = None*, *credentials: Credentials*, *db_specific_attributes: Optional[DatabaseAttributes] = None*, *url_params: Optional[List[Tuple[str, str]]] = None*)

Bases: *CatalogDataSourceBase*

__init__(*, *id: str*, *name: str*, *type: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[str, str]]] = None*, *decoded_parameters: Optional[List[Dict[str, str]]] = None*, *credentials: Credentials*, *db_specific_attributes: Optional[DatabaseAttributes] = None*, *url_params: Optional[List[Tuple[str, str]]] = None*) → None

Method generated by attrs for class CatalogDataSource.

Methods

__init__(*, id, name, type, schema[, url, ...])	Method generated by attrs for class CatalogDataSource.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

url_template
db_vendor
db_specific_attributes
url_params

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

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.CatalogDataSourceBase

class gooddata_sdk.catalog.data_source.entity_model.data_source.**CatalogDataSourceBase**(*, *id: str*, *name: str*, *type: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[str, str]]] = None*, *decoded_parameters: Optional[List[Dict[str, str]]] = None*, *credentials: Credentials*)

Bases: *Base*

__init__(*, *id: str*, *name: str*, *type: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[str, str]]] = None*, *decoded_parameters: Optional[List[Dict[str, str]]] = None*, *credentials:* Credentials) → None

Method generated by attrs for class CatalogDataSourceBase.

Methods

__init__(*, id, name, type, schema[, url, ...])	Method generated by attrs for class CatalogDataSourceBase.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
name
type
schema
url
enable_caching
cache_path
parameters
decoded_parameters
credentials

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

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.CatalogDataSourceBigQuery

class gooddata_sdk.catalog.data_source.entity_model.data_source.**CatalogDataSourceBigQuery**(*, *id: str*, *name: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[s str]]] = None*, *decoded_parameter Optional[List[Dict[s str]]] = None*, *credentials: Credentials*, *db_specific_attril Optional[DatabaseA = None*, *url_params: Optional[List[Tuple[str]]] = None*, *type: str = 'BIG-*

Bases: CatalogDataSource

__init__(*, id: str, name: str, schema: str, url: Optional[str] = None, enable_caching: Optional[bool] = None, cache_path: Optional[List[str]] = None, parameters: Optional[List[Dict[str, str]]] = None, decoded_parameters: Optional[List[Dict[str, str]]] = None, credentials: Credentials, db_specific_attributes: Optional[DatabaseAttributes] = None, url_params: Optional[List[Tuple[str, str]]] = None, type: str = 'BIGQUERY') → None

Method generated by attrs for class CatalogDataSourceBigQuery.

Methods

__init__(*, id, name, schema[, url, ...])	Method generated by attrs for class CatalogDataSourceBigQuery.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

url_template
type

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

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.CatalogDataSourceGreenplum

class gooddata_sdk.catalog.data_source.entity_model.data_source.**CatalogDataSourceGreenplum**(*, *id: str*, *name: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[str, str]]] = None*, *decoded_parameters: Optional[List[Dict[str, str]]] = None*, *credentials: Credentials*, *db_specific_attributes: Optional[DatabaseSpecificAttributes] = None*, *url_params: Optional[List[Tuple[str, str]]] = None*, *type: str = 'GREEN-*

Bases: CatalogDataSourcePostgres

__init__(*, id: str, name: str, schema: str, url: Optional[str] = None, enable_caching: Optional[bool] = None, cache_path: Optional[List[str]] = None, parameters: Optional[List[Dict[str, str]]] = None, decoded_parameters: Optional[List[Dict[str, str]]] = None, credentials: Credentials, db_specific_attributes: Optional[DatabaseAttributes] = None, url_params: Optional[List[Tuple[str, str]]] = None, type: str = 'GREENPLUM', db_vendor: str = 'postgresql') → None

Method generated by attrs for class CatalogDataSourceGreenplum.

Methods

__init__(*, id, name, schema[, url, ...])	Method generated by attrs for class CatalogDataSourceGreenplum.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

url_template
type
db_vendor

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

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.CatalogDataSourcePostgres

class gooddata_sdk.catalog.data_source.entity_model.data_source.**CatalogDataSourcePostgres**(*, *id: str*, *name: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[s str]]] = None*, *decoded_parameter Optional[List[Dict[s str]]] = None*, *credentials: Credentials*, *db_specific_attril Optional[DatabaseA = None*, *url_params: Optional[List[Tuple[str]]] = None*, *type: str = 'POST-*

Bases: CatalogDataSource

__init__(*, id: str, name: str, schema: str, url: Optional[str] = None, enable_caching: Optional[bool] = None, cache_path: Optional[List[str]] = None, parameters: Optional[List[Dict[str, str]]] = None, decoded_parameters: Optional[List[Dict[str, str]]] = None, credentials: Credentials, db_specific_attributes: Optional[DatabaseAttributes] = None, url_params: Optional[List[Tuple[str, str]]] = None, type: str = 'POSTGRESQL') → None

Method generated by attrs for class CatalogDataSourcePostgres.

Methods

__init__(*, id, name, schema[, url, ...])	Method generated by attrs for class CatalogDataSourcePostgres.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

url_template
type

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

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.CatalogDataSourceRedshift

class gooddata_sdk.catalog.data_source.entity_model.data_source.**CatalogDataSourceRedshift**(*, *id: str*, *name: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[s str]]] = None*, *decoded_parameter Optional[List[Dict[s str]]] = None*, *credentials: Credentials*, *db_specific_attril Optional[DatabaseA = None*, *url_params: Optional[List[Tuple[str]]] = None*, *type: str = 'RED-*

Bases: CatalogDataSourcePostgres

__init__(*, id: str, name: str, schema: str, url: Optional[str] = None, enable_caching: Optional[bool] = None, cache_path: Optional[List[str]] = None, parameters: Optional[List[Dict[str, str]]] = None, decoded_parameters: Optional[List[Dict[str, str]]] = None, credentials: Credentials, db_specific_attributes: Optional[DatabaseAttributes] = None, url_params: Optional[List[Tuple[str, str]]] = None, type: str = 'REDSHIFT') → None

Method generated by attrs for class CatalogDataSourceRedshift.

Methods

__init__(*, id, name, schema[, url, ...])	Method generated by attrs for class CatalogDataSourceRedshift.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

url_template
type

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

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.CatalogDataSourceSnowflake

class gooddata_sdk.catalog.data_source.entity_model.data_source.**CatalogDataSourceSnowflake**(*, *id: str*, *name: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[str, str]]] = None*, *decoded_parameters: Optional[List[Dict[str, str]]] = None*, *credentials: Credentials*, *url_params: Optional[List[Tuple[str, str]]] = None*, *type: str = 'SNOWFLAKE'*, *db_specific_attributes: DatabaseAttributes*)

Bases: CatalogDataSource

__init__(*, *id: str*, *name: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[str, str]]] = None*, *decoded_parameters: Optional[List[Dict[str, str]]] = None*, *credentials:* Credentials, *url_params: Optional[List[Tuple[str, str]]] = None*, *type: str = 'SNOWFLAKE'*, *db_specific_attributes:* DatabaseAttributes) → None

Method generated by attrs for class CatalogDataSourceSnowflake.

Methods

__init__(*, id, name, schema[, url, ...])	Method generated by attrs for class CatalogDataSourceSnowflake.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

url_template
type
db_specific_attributes

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

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.CatalogDataSourceVertica

class gooddata_sdk.catalog.data_source.entity_model.data_source.**CatalogDataSourceVertica**(*, *id: str*, *name: str*, *schema: str*, *url: Optional[str] = None*, *enable_caching: Optional[bool] = None*, *cache_path: Optional[List[str]] = None*, *parameters: Optional[List[Dict[st str]]] = None*, *decoded_parameters Optional[List[Dict[st str]]] = None*, *credentials: Credentials*, *db_specific_attribu Optional[DatabaseAtt = None*, *url_params: Optional[List[Tuple[s str]]] = None*, *type: str = 'VER-*

Bases: CatalogDataSourcePostgres

__init__(*, id: str, name: str, schema: str, url: Optional[str] = None, enable_caching: Optional[bool] = None, cache_path: Optional[List[str]] = None, parameters: Optional[List[Dict[str, str]]] = None, decoded_parameters: Optional[List[Dict[str, str]]] = None, credentials: Credentials, db_specific_attributes: Optional[DatabaseAttributes] = None, url_params: Optional[List[Tuple[str, str]]] = None, type: str = 'VERTICA') → None

Method generated by attrs for class CatalogDataSourceVertica.

Methods

__init__(*, id, name, schema[, url, ...])	Method generated by attrs for class CatalogDataSourceVertica.
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.
to_api()	
to_api_patch(data_source_id, attributes)	
to_dict([camel_case])	Converts object into dictionary.

Attributes

url_template
type

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

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.DatabaseAttributes

class gooddata_sdk.catalog.data_source.entity_model.data_source.**DatabaseAttributes**

Bases: `object`

__init__() → None

Method generated by attrs for class DatabaseAttributes.

Methods

__init__()	Method generated by attrs for class DatabaseAttributes.

Attributes

`str_attributes`

gooddata_sdk.catalog.data_source.entity_model.data_source.GreenplumAttributes

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

Bases: *PostgresAttributes*

__init__(*, *host: str*, *db_name: str*, *port: str = '5432'*) → None

Method generated by attrs for class GreenplumAttributes.

Methods

__init__(*, host, db_name[, port])	Method generated by attrs for class GreenplumAttributes.

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'*) → None

Method generated by attrs for class PostgresAttributes.

Methods

__init__(*, host, db_name[, port])	Method generated by attrs for class PostgresAttributes.

Attributes

str_attributes
host
db_name
port

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'*) → None

Method generated by attrs for class RedshiftAttributes.

Methods

__init__(*, host, db_name[, port])	Method generated by attrs for class RedshiftAttributes.

Attributes

str_attributes
port

gooddata_sdk.catalog.data_source.entity_model.data_source.SnowflakeAttributes

class gooddata_sdk.catalog.data_source.entity_model.data_source.**SnowflakeAttributes**(*, *account: str*, *warehouse: str*, *db_name: str*, *port: str = '443'*)

Bases: *DatabaseAttributes*

__init__(*, *account: str*, *warehouse: str*, *db_name: str*, *port: str = '443'*) → None

Method generated by attrs for class SnowflakeAttributes.

Methods

__init__(*, account, warehouse, db_name[, port])	Method generated by attrs for class SnowflakeAttributes.

Attributes

str_attributes
account
warehouse
db_name
port

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'*) → None

Method generated by attrs for class VerticaAttributes.

Methods

__init__(*, host, db_name[, port])	Method generated by attrs for class VerticaAttributes.

Attributes

str_attributes
port

gooddata_sdk.catalog.data_source.service

Classes

CatalogDataSourceService(api_client)	_summary_

gooddata_sdk.catalog.data_source.service.CatalogDataSourceService

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

Bases: *CatalogServiceBase*

summary

Args:
CatalogServiceBase (_type_): _description_

__init__(*api_client:* GoodDataApiClient) → None

Methods

__init__(api_client)	
create_or_update_data_source(data_source)	Pushes the Data Source to the GoodData environment.
data_source_folder(data_source_id, ...)	TODO
delete_data_source(data_source_id)	Delete data source using Data Source id.
generate_logical_model(data_source_id[, ...])	Generate logical data model for a data source.
get_data_source(data_source_id)	Retrieve Data Source entity using data source id.
get_declarative_data_sources()	Retrieve all data sources, including their related physical data model.
get_declarative_pdm(data_source_id)	Retrieve physical data model for a given data source.
get_organization()	
layout_organization_folder(layout_root_path)	
list_data_source_tables(data_source_id)	Lists all the data source tables for a given data source.
list_data_sources()	Lists all data sources.
load_and_put_declarative_data_sources([...])	This method combines load_declarative_data_sources and put_declarative_data_sources
load_and_put_declarative_pdm(data_source_id)	This method combines load_declarative_pdm and put_declarative_pdm methods
load_declarative_data_sources([layout_root_path])	Load declarative data sources layout, which was stored using store_declarative_data_sources.
load_declarative_pdm(data_source_id[, ...])	Load declarative physical data model layout, which was stored using store_declarative_pdm for a given data source.
load_pdm_from_disk([path])	This method is used to load pdm stored to disk using method store_pdm_to_disk.
patch_data_source_attributes(data_source_id, ...)	Applies changes to the specified data source.
put_declarative_data_sources(...[, ...])	Set all data sources, including their related physical data model.
put_declarative_pdm(data_source_id, ...)	Set physical data model for a given data source.
register_upload_notification(data_source_id)	Invalidate cache of your computed reports to force your analytics to be recomputed.
report_warnings(warnings)	
scan_and_put_pdm(data_source_id[, scan_request])	This method combines scan_data_source and put_declarative_pdm methods.
scan_data_source(data_source_id[, ...])	Scan data source specified by its id and optionally by specified scan request.
scan_schemata(data_source_id)	Returns a list of schemas that exist in the database and can be configured in the data source entity.
scan_sql(data_source_id, sql_request)	Analyze SELECT SQL query in a given request.
store_declarative_data_sources([...])	Store data sources layouts in a directory hierarchy.
store_declarative_pdm(data_source_id[, ...])	Store physical data 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.
store_pdm_to_disk(datasource_id[, path])	Store the physical data model layout in the directory for a given data source.
test_data_sources_connection(...[, ...])	Tests connection to declarative data sources.

Attributes

`organization_id`

create_or_update_data_source(*data_source:* CatalogDataSource) → None

Pushes the Data Source to the GoodData environment. Automatically decides, whether to create or update.

Args:

data_source (CatalogDataSource):
Catalog Data Source object

Returns:
None

data_source_folder(*data_source_id: str*, *layout_root_path: Path*) → Path

TODO

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

layout_root_path (Path):
...

Returns:

Path:
Path to the source folder.

delete_data_source(*data_source_id: str*) → None

Delete data source using Data Source id.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

Returns:
None

generate_logical_model(*data_source_id: str*, *generate_ldm_request:* CatalogGenerateLdmRequest = *CatalogGenerateLdmRequest(separator='__', generate_long_ids=None, table_prefix=None, view_prefix=None, primary_label_prefix=None, secondary_label_prefix=None, fact_prefix=None, date_granularities=None, grain_prefix=None, reference_prefix=None, grain_reference_prefix=None, denorm_prefix=None, wdf_prefix='wdf', pdm=None)*) → *CatalogDeclarativeModel*

Generate logical data model for a data source.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

generate_ldm_request (CatalogGenerateLdmRequest, optional):
LDM options. Defaults to CatalogGenerateLdmRequest(separator="__", wdf_prefix="wdf")

Returns:

CatalogDeclarativeModel:
Object Containing declarative Logical Data Model

`get_data_source`(*data_source_id: str*) → *CatalogDataSource*

Retrieve Data Source entity using data source id.

Args:
data_source_id (str): Data Source identification string e.g. "demo"

Returns:
CatalogDataSource: Data Source Object

`get_declarative_data_sources`() → *CatalogDeclarativeDataSources*

Retrieve all data sources, including their related physical data model.

Args:
None

Returns:

CatalogDeclarativeDataSources:
Declarative Data Sources, including physical data model.

`get_declarative_pdm`(*data_source_id: str*) → *CatalogDeclarativeTables*

Retrieve physical data model for a given data source.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

Returns:

CatalogDeclarativeTables:
Physical Data Model object.

`list_data_source_tables`(*data_source_id: str*) → List[*CatalogDataSourceTable*]

Lists all the data source tables for a given data source.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

Returns:

List[CatalogDataSourceTable]:
List of Data Source Table objects

`list_data_sources`() → List[*CatalogDataSource*]

Lists all data sources.

Args:
None

Returns:

List[CatalogDataSource]:
List of all Data Sources in the whole organization.

`load_and_put_declarative_data_sources`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*, *credentials_path: Optional[Path] = None*, *test_data_sources: bool = False*) → None

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

Args:

layout_root_path (Optional[Path], optional):
Path to the root of the layout directory. Defaults to Path.cwd().

credentials_path (Optional[Path], optional):
Path to the credentials. Defaults to Path.cwd().

test_data_sources (bool, optional):
If True, the connection of data sources is tested. Defaults to False.

Returns:
None

`load_and_put_declarative_pdm`(*data_source_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

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

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`load_declarative_data_sources`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeDataSources*

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

Args:

layout_root_path (Optional[Path], optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeDataSources:
Declarative Data Sources object

`load_declarative_pdm`(*data_source_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeTables*

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

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
CatalogDeclarativeTables: Physical Data Model object.

`static load_pdm_from_disk`(*path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeTables*

This method is used to load pdm stored to disk using method store_pdm_to_disk.

Args:

path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeTables:
Physical Data Model object.

`patch_data_source_attributes`(*data_source_id: str*, *attributes: dict*) → None

Applies changes to the specified data source.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

attributes (dict):
A dictionary containing attributes of the data source to be changed.

Returns:
None

`put_declarative_data_sources`(*declarative_data_sources:* CatalogDeclarativeDataSources, *credentials_path: Optional[Path] = None*, *test_data_sources: bool = False*) → None

Set all data sources, including their related physical data model.

Args:

declarative_data_sources (CatalogDeclarativeDataSources):
Declarative Data Source object. Can be retrieved by get_declarative_data_sources.

credentials_path (Optional[Path], optional):
Path to the Credentials. Optional, defaults to None.

test_data_sources (bool, optional):
If True, the connection of data sources is tested. Defaults to False.

Returns:
None

`put_declarative_pdm`(*data_source_id: str*, *declarative_tables:* CatalogDeclarativeTables) → None

Set physical data model for a given data source.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

declarative_tables (CatalogDeclarativeTables):
Physical Data Model object. Can be obtained via get_declarative_pdm.

Returns:
None

`register_upload_notification`(*data_source_id: str*) → None

Invalidate cache of your computed reports to force your analytics to be recomputed.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

Returns:
None

`scan_and_put_pdm`(*data_source_id: str*, *scan_request:* CatalogScanModelRequest = *CatalogScanModelRequest(separator='__', scan_tables=True, scan_views=False, table_prefix=None, view_prefix=None, schemata=None)*) → None

This method combines scan_data_source and put_declarative_pdm methods.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

scan_request (CatalogScanModelRequest, optional):
Options for the Scan Request. Defaults to CatalogScanModelRequest().

Returns:
None

`scan_data_source`(*data_source_id: str*, *scan_request:* CatalogScanModelRequest = *CatalogScanModelRequest(separator='__', scan_tables=True, scan_views=False, table_prefix=None, view_prefix=None, schemata=None)*, *report_warnings: bool = False*) → *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.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

scan_request (CatalogScanModelRequest, optional):
Options for the Scan Request. Defaults to CatalogScanModelRequest().

report_warnings (bool, optional):
Switch to turn on warnings. Defaults to False.

Returns:

CatalogScanResultPdm:
An instance of CatalogScanResultPdm. Containing pdm itself and a list of warnings that occurred during scanning.

scan_schemata(*data_source_id: str*) → 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.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

Returns:

list[str]:
List of schema names for the given data source specified by its id.

scan_sql(*data_source_id: str*, *sql_request:* ScanSqlRequest) → *ScanSqlResponse*

Analyze SELECT SQL query in a given request. Return description of SQL result-set as list of column names with GoodData data types and list of example data returned by SELECT query.

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

sql_request (ScanSqlRequest):
SELECT SQL query to analyze

Returns:

ScanSqlResponse:
SELECT query analysis result

store_declarative_data_sources(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store data sources layouts in a 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

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

store_declarative_pdm(*data_source_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store physical data 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

Args:

data_source_id (str):
Data Source identification string. e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`store_pdm_to_disk`(*datasource_id: str*, *path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store the physical data model layout in the directory for a given data source.
The directory structure below shows the output for the path set to Path("pdm_location"). pdm_location

└── **pdm**
├── table_A.yaml └── table_B.yaml

Args:

datasource_id (str):
Data Source identification string. e.g. "demo"

path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`test_data_sources_connection`(*declarative_data_sources: CatalogDeclarativeDataSources*, *credentials_path: Optional[Path] = None*) → 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".

Args:

declarative_data_sources (CatalogDeclarativeDataSources):
Declarative Data Sources object

credentials_path (Optional[Path], optional):
Path to the credentials. Defaults to None.

Raises:

ValueError:
Check API references for possible errors of data source connections.

Returns:
None

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*) → None

Method generated by attrs for class BasicCredentials.

Methods

__init__(*, username, password)	Method generated by attrs for class BasicCredentials.
client_class()	
create(creds_classes, entity)	
from_api(attributes)	Creates object from entity passed by client class, which represents it as dictionary.
from_dict(data[, camel_case])	Creates object from dictionary.
is_part_of_api(entity)	
to_api()	
to_api_args()	
to_dict([camel_case])	Converts object into dictionary.
validate_instance(creds_classes, instance)	

Attributes

PASSWORD_KEY
TOKEN_KEY
USER_KEY
username
password

classmethod from_api(*attributes: dict[str, Any]*) → *BasicCredentials*

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.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: *Base*

__init__() → None

Method generated by attrs for class Credentials.

Methods

__init__()	Method generated by attrs for class Credentials.
client_class()	
create(creds_classes, entity)	
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.
is_part_of_api(entity)	
to_api()	
to_api_args()	
to_dict([camel_case])	Converts object into dictionary.
validate_instance(creds_classes, instance)	

Attributes

PASSWORD_KEY
TOKEN_KEY
USER_KEY

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.entity.TokenCredentials

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

Bases: *Credentials*

__init__(*, *token: str*) → None

Method generated by attrs for class TokenCredentials.

Methods

__init__(*, token)	Method generated by attrs for class TokenCredentials.
client_class()	
create(creds_classes, entity)	
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.
is_part_of_api(entity)	
to_api()	
to_api_args()	
to_dict([camel_case])	Converts object into dictionary.
validate_instance(creds_classes, instance)	

Attributes

PASSWORD_KEY
TOKEN_KEY
USER_KEY
token

classmethod from_api(*entity: dict[str, Any]*) → *TokenCredentials*

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.entity.TokenCredentialsFromFile

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

Bases: *Credentials*

__init__(*, *file_path: Path*) → None

Method generated by attrs for class TokenCredentialsFromFile.

Methods

__init__(*, file_path)	Method generated by attrs for class TokenCredentialsFromFile.
client_class()	
create(creds_classes, entity)	
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.
is_part_of_api(entity)	
to_api()	
to_api_args()	
to_dict([camel_case])	Converts object into dictionary.
token_from_file(file_path)	
validate_instance(creds_classes, instance)	

Attributes

PASSWORD_KEY
TOKEN_KEY
USER_KEY
file_path
token

classmethod from_api(*entity: dict[str, Any]*) → *TokenCredentialsFromFile*

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

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

__init__(*, id, type)	Method generated by attrs for class CatalogAssigneeIdentifier.
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.
to_api()	
to_dict([camel_case])	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.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

__init__(*, id, type)	Method generated by attrs for class CatalogGrainIdentifier.
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.
to_api()	
to_dict([camel_case])	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.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

__init__(*, id, type)	Method generated by attrs for class CatalogLabelIdentifier.
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.
to_api()	
to_dict([camel_case])	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.CatalogReferenceIdentifier

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

Bases: *Base*

__init__(*, *id: str*) → None

Method generated by attrs for class CatalogReferenceIdentifier.

Methods

__init__(*, id)	Method generated by attrs for class CatalogReferenceIdentifier.
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.
to_api()	
to_dict([camel_case])	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.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

__init__(*, id, type)	Method generated by attrs for class CatalogUserGroupIdentifier.
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.
to_api()	
to_dict([camel_case])	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

__init__(*, id)	Method generated by attrs for class CatalogWorkspaceIdentifier.
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.
to_api()	
to_dict([camel_case])	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*, *attributes:* CatalogOrganizationAttributes)

Bases: *Base*

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

Method generated by attrs for class CatalogOrganization.

Methods

__init__(*, id, attributes)	Method generated by attrs for class CatalogOrganization.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
attributes

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: 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*)

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

__init__(*[, name, hostname, ...])	Method generated by attrs for class CatalogOrganizationAttributes.
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.
to_api()	
to_dict([camel_case])	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:* CatalogOrganization)

Bases: *Base*

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

Method generated by attrs for class CatalogOrganizationDocument.

Methods

__init__(*, data)	Method generated by attrs for class CatalogOrganizationDocument.
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.
to_api([oauth_client_secret])	
to_dict([camel_case])	Converts object into dictionary.

Attributes

data

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

CatalogOrganizationService(api_client)	

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)	Updates the name of the organization.
update_oidc_parameters([...])	Updates OIDC parameters of organization.

Attributes

organization_id

update_name(*name: str*) → None

Updates the name of the organization.

Args:

name (str):
New name of the organization

Returns:
None

update_oidc_parameters(*oauth_issuer_location: Optional[str] = None*, *oauth_client_id: Optional[str] = None*, *oauth_client_secret: Optional[str] = None*) → None

Updates OIDC parameters of organization.

Args:

oauth_issuer_location (Optional[str], optional):
Issuer location. Defaults to None.

oauth_client_id (Optional[str], optional):
Public client identifier. Defaults to None.

oauth_client_secret (Optional[str], optional):
Client secret. Defaults to None.

Returns:
None

Raises:

ValueError:
Parameters were not strictly all none or all string.

gooddata_sdk.catalog.parameter

Classes

CatalogParameter(*, name, value)	

gooddata_sdk.catalog.parameter.CatalogParameter

class gooddata_sdk.catalog.parameter.**CatalogParameter**(*, *name: str*, *value: str*)

Bases: *Base*

__init__(*, *name: str*, *value: str*) → None

Method generated by attrs for class CatalogParameter.

Methods

__init__(*, name, value)	Method generated by attrs for class CatalogParameter.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

name
value

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

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.**CatalogDeclarativeDataSourcePermissi**

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeDataSourcePermission.

Methods

__init__(*, name, assignee)	Method generated by attrs for class CatalogDeclarativeDataSourcePermission.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

name
assignee

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.**CatalogDeclarativeSingleWorkspacePer**

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeSingleWorkspacePermission.

Methods

__init__(*, name, assignee)	Method generated by attrs for class CatalogDeclarativeSingleWorkspacePermission.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

name
assignee

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.CatalogDeclarativeWorkspaceHierarchyPermissi

class gooddata_sdk.catalog.permission.declarative_model.permission.**CatalogDeclarativeWorkspaceHierarchyP**

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeWorkspaceHierarchyPermission.

Methods

__init__(*, name, assignee)	Method generated by attrs for class CatalogDeclarativeWorkspaceHierarchyPermission.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

name
assignee

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.**CatalogDeclarativeWorkspacePermissio**

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeWorkspacePermissions.

Methods

__init__(*[, permissions, hierarchy_permissions])	Method generated by attrs for class CatalogDeclarativeWorkspacePermissions.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

permissions
hierarchy_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.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)	Retrieve current set of permissions of the workspace in a declarative form.
get_organization()	
layout_organization_folder(layout_root_path)	
put_declarative_permissions(workspace_id, ...)	Set effective permissions for the workspace.

Attributes

organization_id

get_declarative_permissions(*workspace_id: str*) → *CatalogDeclarativeWorkspacePermissions*

Retrieve current set of permissions of the workspace in a declarative form.

Args:

workspace_id (str):
Workspace identification string. e.g. “demo”

Returns:

CatalogDeclarativeWorkspacePermissions:
Object containing workspace permissions.

put_declarative_permissions(*workspace_id: str*, *declarative_workspace_permissions:* CatalogDeclarativeWorkspacePermissions) → None

Set effective permissions for the workspace.

Args:

workspace_id (str):
Workspace identification string. e.g. “demo”

declarative_workspace_permissions (CatalogDeclarativeWorkspacePermissions):
Object containing workspace Permissions.

Returns:
None

gooddata_sdk.catalog.setting

Classes

CatalogDeclarativeCustomApplicationSetting(*, ...)	
CatalogDeclarativeSetting(*, id[, content])	

gooddata_sdk.catalog.setting.CatalogDeclarativeCustomApplicationSetting

class gooddata_sdk.catalog.setting.**CatalogDeclarativeCustomApplicationSetting**(*, *id: str*, *content: Dict[str, Any]*, *application_name: str*)

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeCustomApplicationSetting.

Methods

__init__(*, id, content, application_name)	Method generated by attrs for class CatalogDeclarativeCustomApplicationSetting.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
content
application_name

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.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

__init__(*, id[, content])	Method generated by attrs for class CatalogDeclarativeSetting.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
content

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

__init__(*, id[, auth_id, user_groups, settings])	Method generated by attrs for class CatalogDeclarativeUser.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
auth_id
user_groups
settings

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

__init__(*, users)	Method generated by attrs for class CatalogDeclarativeUsers.
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

users

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

CatalogDeclarativeUsersUserGroups(*, users, ...)

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

__init__(*, users, user_groups)	Method generated by attrs for class CatalogDeclarativeUsersUserGroups.
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

users
user_groups

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

CatalogDeclarativeUserGroup(*, id[, parents])
CatalogDeclarativeUserGroups(*[, user_groups])

gooddata_sdk.catalog.user.declarative_model.user_group.CatalogDeclarativeUserGroup

class gooddata_sdk.catalog.user.declarative_model.user_group.**CatalogDeclarativeUserGroup**(*, *id: str*, *parents: Optional[List[Catalog* = *None*)

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeUserGroup.

Methods

__init__(*, id[, parents])	Method generated by attrs for class CatalogDeclarativeUserGroup.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
parents

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[CatalogDecl = NOTHING*)

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeUserGroups.

Methods

__init__(*[, user_groups])	Method generated by attrs for class CatalogDeclarativeUserGroups.
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

user_groups

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: Optional[CatalogUserRelationships] = None*)

Bases: *Base*

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

Method generated by attrs for class CatalogUser.

Methods

__init__(*, id[, attributes, relationships])	Method generated by attrs for class CatalogUser.
add_user_group(user_group)	
add_user_groups(user_groups)	
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.
init(user_id[, authentication_id, ...])	
remove_user_groups()	
replace_user_groups(user_groups)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

user_groups
id
attributes
relationships

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

__init__(*[, authentication_id])	Method generated by attrs for class CatalogUserAttributes.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

authentication_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.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

__init__(*, data)	Method generated by attrs for class CatalogUserDocument.
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.
init(user_id[, authentication_id, ...])	
to_api()	
to_dict([camel_case])	Converts object into dictionary.
update_user([authentication_id, user_group_ids])	

Attributes

data

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: List[CatalogUserGroup] = NOTHING*)

Bases: *Base*

__init__(*, *data: List[CatalogUserGroup] = NOTHING*) → None

Method generated by attrs for class CatalogUserGroupsData.

Methods

__init__(*[, data])	Method generated by attrs for class CatalogUserGroupsData.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

data

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

__init__(*[, user_groups])	Method generated by attrs for class CatalogUserRelationships.
add_user_groups(user_groups)	
client_class()	
create_user_relationships(user_group_ids)	
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.
replace_user_groups(user_groups)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

user_groups

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

CatalogUserGroup(*, id[, relationships])
CatalogUserGroupDocument(*, data)
CatalogUserGroupParents(*[, data])
CatalogUserGroupRelationships(*[, parents])

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

__init__(*, id[, relationships])	Method generated by attrs for class CatalogUserGroup.
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.
init(user_group_id[, user_group_parent_ids])	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

get_parents
id
relationships

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

__init__(*, data)	Method generated by attrs for class CatalogUserGroupDocument.
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.
init(user_group_id[, user_group_parent_ids])	
to_api()	
to_dict([camel_case])	Converts object into dictionary.
update_user_group([user_group_parents_id])	

Attributes

data

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[*CatalogUserGroup *= None*)

Bases: *Base*

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

Method generated by attrs for class CatalogUserGroupParents.

Methods

__init__(*[, data])	Method generated by attrs for class CatalogUserGroupParents.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

get_parents
data

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[*CatalogUserGro *= None*)

Bases: *Base*

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

Method generated by attrs for class CatalogUserGroupRelationships.

Methods

__init__(*[, parents])	Method generated by attrs for class CatalogUserGroupRelationships.
client_class()	
create_user_group_relationships(...)	
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

get_parents
parents

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

CatalogUserService(api_client)

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)	Creates a new user or overwrites an existing user.
create_or_update_user_group(user_group)	Create a new user group or overwrite an existing user group.
delete_user(user_id)	Delete User using User id.
delete_user_group(user_group_id)	Delete User Group using User Group id.
get_declarative_user_groups()	Retrieve all user groups in a declarative form.
get_declarative_users()	Retrieve all users in a declarative form.
get_declarative_users_user_groups()	Retrieves all users and user groups in a declarative form.
get_organization()	
get_user(user_id)	Get an individual user using User id.
get_user_group(user_group_id)	Get an individual user group using user group id.
layout_organization_folder(layout_root_path)	
list_user_groups()	Get a list of all existing user groups.
list_users()	Get a list of all existing users.
load_and_put_declarative_user_groups([...])	This method combines load_declarative_users and put_declarative_users
load_and_put_declarative_users([...])	This method combines load_declarative_users and put_declarative_users
load_and_put_declarative_users_user_groups([...])	This method combines load_declarative_users and put_declarative_users_user_groups
load_declarative_user_groups([layout_root_path])	Load declarative users groups layout, which was stored using store_declarative_user_groups.
load_declarative_users([layout_root_path])	Load declarative users layout, which was stored using store_declarative_users.
load_declarative_users_user_groups([...])	Load declarative users and user groups layout, which was stored using store_declarative_users_user_groups.
put_declarative_user_groups(user_groups)	Set all user groups eventually with their parents.
put_declarative_users(users)	Set all users and their authentication properties.
put_declarative_users_user_groups(...)	Set all users and user groups.
store_declarative_user_groups([layout_root_path])	Stores all the user groups in a directory hierarchy.
store_declarative_users([layout_root_path])	Store users in directory hierarchy.
store_declarative_users_user_groups([...])	Stores all the users and user groups in a directory hierarchy.

Attributes

`organization_id`

`create_or_update_user`(*user:* CatalogUser) → None

Creates a new user or overwrites an existing user.

Args:

user (CatalogUser):
User entity object.

Returns:
None

`create_or_update_user_group`(*user_group:* CatalogUserGroup) → None

Create a new user group or overwrite an existing user group.

Args:

user_group (CatalogUserGroup):
UserGroup entity object.

Returns:
None

`delete_user`(*user_id: str*) → None

Delete User using User id.

Args:

user_id (str):
User identification string. e.g. “123”

Returns:
None

`delete_user_group`(*user_group_id: str*) → None

Delete User Group using User Group id.

Args:

user_group_id (str):
User Group identification string. e.g. “123”

Returns:
None

`get_declarative_user_groups`() → *CatalogDeclarativeUserGroups*

Retrieve all user groups in a declarative form.

Args:
None

Returns:

CatalogDeclarativeUserGroups:
Declarative User Groups object.

`get_declarative_users()` → *CatalogDeclarativeUsers*

Retrieve all users in a declarative form.

Args:
None

Returns:

CatalogDeclarativeUsers:
Declarative Users object.

`get_declarative_users_user_groups()` → *CatalogDeclarativeUsersUserGroups*

Retrieves all users and user groups in a declarative form.

Args:
None

Returns:

CatalogDeclarativeUsersUserGroups:
Declarative Users and User Groups object.

`get_user(user_id: str)` → *CatalogUser*

Get an individual user using User id.

Args:

user_id (str):
User identification string. e.g. "123"

Returns:

CatalogUser:
User entity object.

`get_user_group(user_group_id: str)` → *CatalogUserGroup*

Get an individual user group using user group id.

Args:

user_group_id (str):
User Group identification string. e.g. "123"

Returns:

CatalogUserGroup:
UserGroup entity object.

`list_user_groups()` → List[*CatalogUserGroup*]

Get a list of all existing user groups.

Args:
None

Returns:

List[CatalogUserGroup]:
List of all User groups as UserGroup entity object.

`list_users()` → List[*CatalogUser*]

Get a list of all existing users.

Args:
None

Returns:

List[CatalogUser]:
List of all Users as User entity objects.

`load_and_put_declarative_user_groups`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

This method combines load_declarative_users and put_declarative_users
methods to load and set layouts stored using store_declarative_users.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:
None

`load_and_put_declarative_users`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

This method combines load_declarative_users and put_declarative_users
methods to load and set layouts stored using store_declarative_users.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:
None

`load_and_put_declarative_users_user_groups`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

This method combines load_declarative_users and put_declarative_users_user_groups
methods to load and set layouts stored using store_declarative_users_user_groups.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:
None

`load_declarative_user_groups`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeUserGroups*

Load declarative users groups layout, which was stored using store_declarative_user_groups.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:

CatalogDeclarativeUserGroups:
Declarative User Groups object.

`load_declarative_users`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeUsers*

Load declarative users layout, which was stored using store_declarative_users.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:

CatalogDeclarativeUsers:
Declarative Users object, incuding authetication properties.

`load_declarative_users_user_groups`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeUsersUserGroups*

Load declarative users and user groups layout, which was stored using store_declarative_users_user_groups.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:

CatalogDeclarativeUsersUserGroups:
Declarative Users and User Groups object.

`put_declarative_user_groups`(*user_groups:* CatalogDeclarativeUserGroups) → None

Set all user groups eventually with their parents.

Args:

user_groups (CatalogDeclarativeUserGroups):
Declarative User Groups object.

Returns:
None

`put_declarative_users`(*users:* CatalogDeclarativeUsers) → None

Set all users and their authentication properties.

Args:

users (CatalogDeclarativeUsers):
Declarative Users object, incuding authetication properties.

Returns:
None

put_declarative_users_user_groups(*users_user_groups: CatalogDeclarativeUsersUserGroups*) → None

Set all users and user groups.

Args:

users_user_groups (CatalogDeclarativeUsersUserGroups):
Declarative Users and User Groups object.

Returns:
None

store_declarative_user_groups(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Stores all the user groups in a directory hierarchy.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:
None

store_declarative_users(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store users in directory hierarchy. Directly from server.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:
None

store_declarative_users_user_groups(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Stores all the users and user groups in a directory hierarchy.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory.. Defaults to Path.cwd().

Returns:
None

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:* GoodDataApiClient)

Bases: *CatalogServiceBase*

__init__(*api_client:* GoodDataApiClient) → None

Methods

__init__(api_client)	
compute_valid_objects(workspace_id, ctx)	Returns attributes, facts, and metrics which are valid to add to a context that already contains some entities from the semantic model.
get_attributes_catalog(workspace_id)	Retrieve all attributes in a given workspace.
get_declarative_analytics_model(workspace_id)	Retrieves declarative analytics model.
get_declarative_ldm(workspace_id)	Retrieve a logical model layout.
get_dependent_entities_graph(workspace_id)	There are dependencies among all catalog objects, the chain is the following: *fact/attribute/label* → *dataset* → *metric* → *visualization* → *dashboard* Some steps can be skipped, e.g.
get_dependent_entities_graph_from_entry_points(...)	Extends get_dependent_entities_graph with the entry point from which the graph is created.
get_facts_catalog(workspace_id)	Retrieve all facts in a given workspace.
get_full_catalog(workspace_id)	Retrieves catalog for a workspace.
get_labels_catalog(workspace_id)	Retrieve all labels in a given workspace.
get_metrics_catalog(workspace_id)	Retrieve all Metrics in a given workspace.
get_organization()	
layout_organization_folder(layout_root_path)	
layout_workspace_folder(workspace_id, ...)	Ties the LDM or Analytics Model to the Organization and workspaces in the store methods.
load_analytics_model_from_disk([path])	Loads the analytics model, which was stored using store_analytics_model_to_disk.
load_and_put_declarative_analytics_model(...)	This method combines load_declarative_analytics_model and put_analytics_model methods to load and set layouts stored using store_declarative_analytics_model.
load_and_put_declarative_ldm(workspace_id[, ...])	This method combines load_declarative_ldm and put_declarative_ldm methods to load and set layouts stored using store_declarative_ldm.
load_declarative_analytics_model(workspace_id)	Loads the declarative analytics model, which was stored using store_declarative_analytics_model.
load_declarative_ldm(workspace_id[, ...])	Load declarative Logical Data Model, which was stored using store_declarative_workspaces
load_ldm_from_disk([path])	Loads the Logical Data Model, which was stored using store_ldm_to_disk.
put_declarative_analytics_model(...)	Sets the declarative analytics model for a given workspace.
put_declarative_ldm(workspace_id, ldm[, ...])	Set declarative logical data model for a given workspace.
store_analytics_model_to_disk(workspace_id)	Store analytics model for a given workspace in directory hierarchy.This method does not tie the declarative
store_declarative_analytics_model(workspace_id)	Store declarative analytics model for a given workspace in directory hierarchy.
store_declarative_ldm(workspace_id[, ...])	Store declarative logical data model for a given workspace in directory hierarchy.
store_ldm_to_disk(workspace_id[, path])	Store declarative logical data model for a given workspace in directory hierarchy.

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.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

ctx (ValidObjectsInputType):
items already in context. you can specify context in one of the following ways:

Returns:

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

get_attributes_catalog(*workspace_id: str*) → list[*CatalogAttribute*]

Retrieve all attributes in a given workspace.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

list[CatalogAttribute]:
List of all attributes in a given workspace.

get_declarative_analytics_model(*workspace_id: str*) → *CatalogDeclarativeAnalytics*

Retrieves declarative analytics model. The model is tied to the workspace and organization.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

CatalogDeclarativeAnalytics:
Object Containing declarative Analytical Model

get_declarative_ldm(*workspace_id: str*) → *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.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

CatalogDeclarativeModel:
Object Containing declarative Logical Data Model.

`get_dependent_entities_graph`(*workspace_id: str*) → *CatalogDependentEntitiesResponse*

There are dependencies among all catalog objects, the chain is the following: *fact/attribute/label* → *dataset* → *metric* → *visualization* → *dashboard* Some steps can be skipped, e.g. *fact* → *visualization* We do not support *table* → *dataset* dependency yet.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

CatalogDependentEntitiesResponse:
Dependent entities graph containing nodes and edges.

`get_dependent_entities_graph_from_entry_points`(*workspace_id: str*, *dependent_entities_request:* CatalogDependentEntitiesRequest) → *CatalogDependentEntitiesResponse*

Extends get_dependent_entities_graph with the entry point from which the graph is created.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

dependent_entities_request (CatalogDependentEntitiesRequest):
Entry points for the dependent entities graph

Returns:

CatalogDependentEntitiesResponse:
Dependent entities graph containing nodes and edges.

`get_facts_catalog`(*workspace_id: str*) → list[*CatalogFact*]

Retrieve all facts in a given workspace.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

list[CatalogFact]:
List of all facts in a given workspace.

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

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

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

CatalogWorkspaceContent:
Object containing all data sets and metrics.

`get_labels_catalog`(*workspace_id: str*) → list[*CatalogLabel*]

Retrieve all labels in a given workspace.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

list[CatalogLabel]:
List of all labels in a given workspace.

`get_metrics_catalog`(*workspace_id: str*) → list[*CatalogMetric*]

Retrieve all Metrics in a given workspace.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

Returns:

list[CatalogMetric]:
List of all metrics in a given workspace.

`layout_workspace_folder`(*workspace_id: str*, *layout_root_path: Path*) → Path

Ties the LDM or Analytics Model to the Organization and workspaces in the store methods.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

Path:
Path to the root of the layout directory for store methods.

`static load_analytics_model_from_disk`(*path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeAnalytics*

Loads the analytics model, which was stored using store_analytics_model_to_disk.

Args:

path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeAnalytics:
Object Containing declarative Analytical Model

`load_and_put_declarative_analytics_model`(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

This method combines load_declarative_analytics_model and put_analytics_model methods to load and set layouts stored using store_declarative_analytics_model.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`load_and_put_declarative_ldm`(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*, *validator: Optional[DataSourceValidator] = None*) → None

This method combines load_declarative_ldm and put_declarative_ldm methods to load and set layouts stored using store_declarative_ldm.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

validator (Optional[DataSourceValidator], optional):
Object that manages validation, whether each data_source_id in LDM corresponds to existing data source. Defaults to None.

Returns:
None

`load_declarative_analytics_model`(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeAnalytics*

Loads the declarative analytics model, which was stored using store_declarative_analytics_model.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeAnalytics:
Object Containing declarative Analytical Model

`load_declarative_ldm`(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeModel*

Load declarative Logical Data Model, which was stored using store_declarative_workspaces

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeModel:
Object Containing declarative Logical Data Model

`static load_ldm_from_disk`(*path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeModel*

Loads the Logical Data Model, which was stored using store_ldm_to_disk.

Args:

path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeModel:
Object Containing declarative Logical Data Model.

`put_declarative_analytics_model`(*workspace_id: str*, *analytics_model:* CatalogDeclarativeAnalytics) → None

Sets the declarative analytics model for a given workspace.

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

analytics_model (CatalogDeclarativeAnalytics):
Object Containing declarative Analytical Model

Returns:
None

`put_declarative_ldm`(*workspace_id: str*, *ldm:* CatalogDeclarativeModel, *validator: Optional[*DataSourceValidator*] = None*) → None

Set declarative logical data model for a given workspace.

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

ldm (CatalogDeclarativeModel):
Object Containing declarative Logical Data Model

validator (Optional[DataSourceValidator], optional):
Object that manages validation, whether each data_source_id in LDM corresponds to existing data source. Defaults to None.

Returns:
None

store_analytics_model_to_disk(*workspace_id: str*, *path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store analytics model for a given workspace in directory hierarchy.This method does not tie the declarative
analytics model to the workspace and organization, thus it is recommended for migration between workspaces. If you want to migrate analytics model between workspaces, use store_analytics_model_to_disk.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

store_declarative_analytics_model(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store declarative analytics model for a given workspace in directory hierarchy.
This method ties the declarative analytics model to the workspace and organization, thus it is recommended for backups. If you want to move declarative analytics model between workspaces or organizations, use store_analytics_model_to_disk.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

store_declarative_ldm(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store declarative logical data model for a given workspace in directory hierarchy.
This method ties the LDM to the workspace and organization, thus it is recommended for backups. If you want to move LDM between workspaces or organizations, use store_ldm_to_disk.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

store_ldm_to_disk(*workspace_id: str*, *path: Path* = *PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store declarative logical data model for a given workspace in directory hierarchy.

This method does not tie the LDM to the workspace and organization, thus it is recommended for migration between organizations. If you want to backup LDM use store_declarative_ldm.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

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.workspace.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.CatalogAnalytic

class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.**Catalo**

Bases: *Base*

__init__(*, *id: str*) → None

Method generated by attrs for class CatalogAnalyticsBase.

Methods

__init__(*, id)	Method generated by attrs for class CatalogAnalyticsBase.
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(analytics_file)	
store_to_disk(analytics_folder)	
to_api()	
to_dict([camel_case])	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.workspace.declarative_model.workspace.analytics_model.analytics_model.CatalogDeclarat

class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.**Catalo**

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

__init__(*, id, title, content[, ...])	Method generated by attrs for class CatalogDeclarativeAnalyticalDashboard.
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(analytics_file)	
store_to_disk(analytics_folder)	
to_api()	
to_dict([camel_case])	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.CatalogDeclarat

class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.**Catalo**

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeAnalytics.

Methods

__init__(*[, analytics])	Method generated by attrs for class CatalogDeclarativeAnalytics.
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)	
store_to_disk(workspace_folder)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

analytics

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.CatalogDeclarat

class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.**Catalo**

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

__init__(*[, analytical_dashboards, ...])	Method generated by attrs for class CatalogDeclarativeAnalyticsLayer.
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.
get_analytical_dashboards_folder(...)	
get_analytics_model_folder(workspace_folder)	
get_dashboard_plugins_folder(...)	
get_filter_contexts_folder(...)	
get_metrics_folder(analytics_model_folder)	
get_visualization_objects_folder(...)	
load_from_disk(workspace_folder)	
store_to_disk(workspace_folder)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

analytical_dashboards
dashboard_plugins
filter_contexts
metrics
visualization_objects

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.CatalogDeclarat

class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.**Catalo**

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

__init__(*, id, title, content[, ...])	Method generated by attrs for class CatalogDeclarativeDashboardPlugin.
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(analytics_file)	
store_to_disk(analytics_folder)	
to_api()	
to_dict([camel_case])	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.CatalogDeclarat

class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.**Catalo**

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 CatalogDeclarativeFilterContext.

Methods

__init__(*, id, title, content[, ...])	Method generated by attrs for class CatalogDeclarativeFilterContext.
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(analytics_file)	
store_to_disk(analytics_folder)	
to_api()	
to_dict([camel_case])	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.CatalogDeclarat

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

__init__(*, id, title, content[, ...])	Method generated by attrs for class CatalogDeclarativeMetric.
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(analytics_file)	
store_to_disk(analytics_folder)	
to_api()	
to_dict([camel_case])	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.CatalogDeclarat

class gooddata_sdk.catalog.workspace.declarative_model.workspace.analytics_model.analytics_model.**Catalo**

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

__init__(*, id, title, content[, ...])	Method generated by attrs for class CatalogDeclarativeVisualizationObject.
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(analytics_file)	
store_to_disk(analytics_folder)	
to_api()	
to_dict([camel_case])	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.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, ...)
CatalogDeclarativeDatasetSql(*, statement, ...)
CatalogDeclarativeFact(*, id, title, ...[, ...])
CatalogDeclarativeLabel(*, id, title, ...[, ...])
CatalogDeclarativeReference(*, identifier, ...)

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

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

Bases: *Base*

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

Method generated by attrs for class CatalogDataSourceTableIdentifier.

Methods

__init__(*, id, data_source_id)	Method generated by attrs for class CatalogDataSourceTableIdentifier.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
data_source_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.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*]*, *source_column_data_type: Optional[str] = None*, *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

__init__(*, id, title, source_column, labels)	Method generated by attrs for class CatalogDeclarativeAttribute.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
title
source_column
labels
source_column_data_type
default_view
sort_column
sort_direction
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, 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, sql: Optional[CatalogDeclarativeDatasetSql] = None, tags: Optional[List[str]] = None, workspace_data_filter_columns: Optional[List[str]] = None) → None

Method generated by attrs for class CatalogDeclarativeDataset.

Methods

__init__(*, id, title, grain, references[, ...])	Method generated by attrs for class CatalogDeclarativeDataset.
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(dataset_file)	
store_to_disk(datasets_folder)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
title
grain
references
description
attributes
facts
data_source_table_id
sql
tags
workspace_data_filter_columns

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__(*, *statement: str*, *data_source_id: str*) → None

Method generated by attrs for class CatalogDeclarativeDatasetSql.

Methods

__init__(*, statement, data_source_id)	Method generated by attrs for class CatalogDeclarativeDatasetSql.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

statement
data_source_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.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*, *source_column_data_type: Optional[str] = None*, *description: Optional[str] = None*, *tags: Optional[List[str]] = None*) → None

Method generated by attrs for class CatalogDeclarativeFact.

Methods

__init__(*, id, title, source_column[, ...])	Method generated by attrs for class CatalogDeclarativeFact.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
title
source_column
source_column_data_type
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*, *source_column_data_type: Optional[str] = None*, *description: Optional[str] = None*, *tags: Optional[List[str]] = None*, *value_type: Optional[str] = None*) → None

Method generated by attrs for class CatalogDeclarativeLabel.

Methods

__init__(*, id, title, source_column[, ...])	Method generated by attrs for class CatalogDeclarativeLabel.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
title
source_column
source_column_data_type
description
tags
value_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.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, *multivalue: bool*, *source_columns: List[str]*, *source_column_data_types: Optional[List[str]] = None*) → None

Method generated by attrs for class CatalogDeclarativeReference.

Methods

__init__(*, identifier, multivalue, ...[, ...])	Method generated by attrs for class CatalogDeclarativeReference.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

identifier
multivalue
source_columns
source_column_data_types

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

__init__(*, id, title, ...[, description, tags])	Method generated by attrs for class CatalogDeclarativeDateDataset.
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(date_instance_file)	
store_to_disk(date_instances_folder)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
title
granularities_formatting
granularities
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.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

__init__(*, title_base, title_pattern)	Method generated by attrs for class CatalogGranularitiesFormatting.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

title_base
title_pattern

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

__init__(*[, datasets, date_instances])	Method generated by attrs for class CatalogDeclarativeLdm.
change_tables_columns_case([upper_case])	Change case (to lower/upper-case) of all physical objects mapped in the LDM.
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.
get_datasets_folder(ldm_folder)	
get_date_instances_folder(ldm_folder)	
get_ldm_folder(workspace_folder)	
load_from_disk(workspace_folder)	
modify_mapped_data_source(data_source_mapping)	LDM contains data source ID - is mapped to this data source.
store_to_disk(workspace_folder)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

datasets
date_instances

change_tables_columns_case(*upper_case: Optional[bool] = None*) → *CatalogDeclarativeLdm*

Change case (to lower/upper-case) of all physical objects mapped in the LDM. Namely mapped table names and column names. Default is to change everything to upper-case. This is handy if you migrate e.g. from PostgreSQL to Snowflake, which is the only DB having upper-case as default. Instead of enclosing all (lower-cased) object names in all DDLs during the migration, you can use this function to change the case in GoodData LDM. If you specify upper-case=False, the function changes the case to lower-case (e.g. migration from Snowflake back to PostgreSQL).

Examples can be found in the DOC of modify_mapped_data_source() method.

Args:

upper_case (bool):
If True, all tables/columns names are changes to upper-case, otherwise to lower-case. If None, noop.

- helps to chaining approach, devs do not have to implement IFs if one of inputs in the chaining is optional.

Returns:
self

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.

modify_mapped_data_source(*data_source_mapping: Optional[dict]*) → *CatalogDeclarativeLdm*

LDM contains data source ID - is mapped to this data source. You may decide to migrate to different data source containing the same physical data model (e.g. change the DB engine, but keep the model). This function helps you to replace any set of data source IDs with new set of IDs (ready for multiple DS per workspace).

Example: ``` data_source_mapping = {"postgresql": "snowflake"} ldm = sdk.catalog_workspace_content.get_declarative_ldm(workspace_id) ldm.modify_mapped_data_source(data_source_mapping) # When migrating to Snowflake, we need to change the case of table/column names as well ldm.change_tables_columns_case(upper_case=True) sdk.catalog_workspace_content.put_declarative_ldm(workspace_id, ldm)

Chaining approach is also possible: ``` sdk.catalog_workspace_content.put_declarative_ldm(

workspace_id, sdk.catalog_workspace_content.get_declarative_ldm(workspace_id)

.modify_mapped_data_source(data_source_mapping).change_tables_columns_case(upper_case=True)

)

Args:

data_source_mapping (dict):
Key value pairs representing which DS(key) should be replaced by which DS(value). If mapping is empty, noop

- helps to chaining approach, devs do not have to implement IFs if one of inputs in the chaining is optional.

Returns:
self

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.**CatalogDeclarativeMo**

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)	
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*, *layout_organization_folder: 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*, *custom_application_settings: List[CatalogDeclarativeCustomApplicationSetting] = NOTHING*) → None

Method generated by attrs for class CatalogDeclarativeWorkspace.

Methods

__init__(*, id, name[, model, parent, ...])	Method generated by attrs for class CatalogDeclarativeWorkspace.
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(workspaces_folder, workspace_id)	
store_to_disk(workspaces_folder)	
to_api([include_nested_structures])	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
name
model
parent
permissions
hierarchy_permissions
early_access
settings
custom_application_settings

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.CatalogDeclarativeWorkspaceDataFilt

class gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.**CatalogDeclarativeWorkspaceDa**

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

__init__(*, id, title, column_name, ...[, ...])	Method generated by attrs for class CatalogDeclarativeWorkspaceDataFilter.
client_class()	
from_api(entity)	Creates object from entity passed by client class, which represents it as dictionary.
from_dict(data[, camel_case])	**param data** Data loaded for example from the file.
load_from_disk(workspaces_data_filter_file)	
store_to_disk(workspaces_data_filters_folder)	
to_api()	
to_dict([camel_case])	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.CatalogDeclarativeWorkspaceDataFilt

class gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.**CatalogDeclarativeWorkspaceDa**

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

__init__(*, id, title, filter_values, workspace)	Method generated by attrs for class CatalogDeclarativeWorkspaceDataFilterSetting.
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.
to_api()	
to_dict([camel_case])	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.CatalogDeclarativeWorkspaceDataFilt

class gooddata_sdk.catalog.workspace.declarative_model.workspace.workspace.**CatalogDeclarativeWorkspaceDa**

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.**CatalogDeclarativeWorkspaceM**

Bases: *Base*

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

Method generated by attrs for class CatalogDeclarativeWorkspaceModel.

Methods

__init__(*[, ldm, analytics])	Method generated by attrs for class CatalogDeclarativeWorkspaceModel.
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)	
store_to_disk(workspace_folder)	
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

ldm
analytics

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

__init__(*, workspaces, workspace_data_filters)	Method generated by attrs for class CatalogDeclarativeWorkspaces.
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.
workspace_data_filters_folder(...)	
workspaces_folder(layout_organization_folder)	

Attributes

workspaces
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.entity_model

Modules

gooddata_sdk.catalog.workspace.entity_model.content_objects
gooddata_sdk.catalog.workspace.entity_model.graph_objects
gooddata_sdk.catalog.workspace.entity_model.workspace

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[*CatalogLabe

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]*, *attributes: list[*CatalogAttribu*facts: list[*CatalogFact*]*)

Bases: *CatalogEntity*

__init__(*entity: dict[str, Any]*, *attributes: list[*CatalogAttribute*]*, *facts: list[*CatalogFact*]*) → None

Methods

__init__(entity, attributes, facts)	
filter_dataset(valid_objects)	Filters dataset so that it contains only attributes and facts that are part of the provided valid objects structure.
find_label_attribute(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(*[, identifiers])
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[
=
NOT
ING
edge
List[
=
NOT
ING

Bases: *Base*

__init__(*, *nodes: List[*CatalogDependentEntitiesNode*] = NOTHING*, *edges: List[List[*CatalogEntityIdentifier*]] = NOTHING*) → None

Method generated by attrs for class CatalogDependentEntitiesGraph.

Methods

__init__(*[, nodes, edges])	Method generated by attrs for class CatalogDependentEntitiesGraph.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

nodes
edges

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

__init__(*, id, type[, title])	Method generated by attrs for class CatalogDependentEntitiesNode.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

id
type
title

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**(*,
id
ti
fi
L
=
N
IN

Bases: *Base*

__init__(*, *identifiers: List[*CatalogEntityIdentifier*] = NOTHING*) → None

Method generated by attrs for class CatalogDependentEntitiesRequest.

Methods

__init__(*[, identifiers])	Method generated by attrs for class CatalogDependentEntitiesRequest.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

identifiers

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

__init__(*, graph)	Method generated by attrs for class CatalogDependentEntitiesResponse.
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.
to_api()	
to_dict([camel_case])	Converts object into dictionary.

Attributes

graph

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

__init__(*, id, type)	Method generated by attrs for class CatalogEntityIdentifier.
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.
to_api()	
to_dict([camel_case])	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.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: functools.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)	
catalog_with_valid_objects(ctx)	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(...)	
find_label_attribute(id_obj)	Get attribute by label id.
get_dataset(dataset_id)	Gets dataset by id.
get_metric(metric_id)	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

__init__(api_client)	
clone_workspace(source_workspace_id[, ...])	Clone workspace from existing workspace.
create_or_update(workspace)	Create a new workspace or overwrite an existing workspace with the same id.
delete_workspace(workspace_id)	Delete a workspace with all its content - logical model and analytics model.
get_declarative_workspace(workspace_id)	Retrieve a workspace layout.
get_declarative_workspace_data_filters()	Retrieve a workspace data filers layout.
get_declarative_workspaces()	Get all workspaces in the current organization in a declarative form.
get_organization()	
get_workspace(workspace_id)	Get an individual workspace.
layout_organization_folder(layout_root_path)	
list_workspaces()	Returns a list of all workspaces in current organization
load_and_put_declarative_workspace(workspace_id)	This method combines load_declarative_workspace and put_declarative_workspace methods to load and set layouts stored using store_declarative_workspace.
load_and_put_declarative_workspace_data_filters([...])	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.
load_and_put_declarative_workspaces([...])	This method combines load_declarative_workspaces and put_declarative_workspaces methods to load and set layouts stored using store_declarative_workspaces.
load_declarative_workspace(workspace_id[, ...])	Load declarative workspaces layout, which was stored using store_declarative_workspace.
load_declarative_workspace_data_filters([...])	Loads workspace data filters layout, which was stored using store_declarative_workspace_data_filters.
load_declarative_workspaces([layout_root_path])	Load declarative workspaces layout, which was stored using store_declarative_workspaces
put_declarative_workspace(workspace_id, ...)	Set a workspace layout.
put_declarative_workspace_data_filters(...)	Set workspace data filers layout.
put_declarative_workspaces(workspace)	Set layout of all workspaces and their hierarchy.
store_declarative_workspace(workspace_id[, ...])	Store workspace layout in a directory hierarchy.
store_declarative_workspace_data_filters([...])	Store workspace data filters layout in a directory hierarchy.
store_declarative_workspaces([layout_root_path])	Stores declarative workspaces in a given path, as folder hierarchy.

Attributes

`organization_id`

`clone_workspace`(*source_workspace_id: str*, *target_workspace_id: Optional[str] = None*, *target_workspace_name: Optional[str] = None*, *overwrite_existing: Optional[bool] = None*, *data_source_mapping: Optional[dict] = None*, *upper_case: Optional[bool] = True*) → None

Clone workspace from existing workspace. Clones complete workspace content - LDM, ADM, permissions.

If the target workspace already exists, it's content is overwritten. This can be useful when testing changes in the clone

- once you are satisfied, you can clone it back to the origin workspace.

For the safety, you have to enforce this behavior by the dedicated input argument *overwrite_existing*.

Beware of workspace data filters - after the clone you have to set WDF value for the new workspace.

Args:

source_workspace_id (str):
Source workspace ID, from which we wanna create a clone

target_workspace_id (str):
Target workspace ID, where we wanna clone the source workspace Optional, if empty, we generate <source_workspace_id>_clone

target_workspace_name (str):
Target workspace name Optional, if empty, we generate <source_workspace_name> (Clone)

overwrite_existing (bool):
Overwrite existing workspace.

data_source_mapping (dict):
Optional, allows users to map LDM to different data source ID

upper_case (bool):
Optional, allows users to change the case of all physical object IDs (table names, columns names) True changes it to upper-case, False to lower-case, None(default) is noop Useful when migrating to Snowflake, which is the only DB with upper-case default.

Returns:
None

`create_or_update`(*workspace: CatalogWorkspace*) → None

Create a new workspace or overwrite an existing workspace with the same id.

Args:

workspace (CatalogWorkspace):
Catalog Workspace object to be created or updated.

Returns:
None

Raises:
ValueError: Workspace parent can not be updated.

`delete_workspace`(*workspace_id: str*) → None

Delete a workspace with all its content - logical model and analytics model.

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

Returns:
None

Raises:

ValueError:
Workspace does not exist.

ValueError:
Workspace is a parent of a workspace.

`get_declarative_workspace`(*workspace_id: str*) → *CatalogDeclarativeWorkspaceModel*

Retrieve a workspace layout.

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

Returns:

CatalogDeclarativeWorkspaceModel:
Object Containing declarative Logical Data Model and declarative Analytical Model.

`get_declarative_workspace_data_filters`() → *CatalogDeclarativeWorkspaceDataFilters*

Retrieve a workspace data filers layout.

Args:
None

Returns:

CatalogDeclarativeWorkspaceDataFilters:
Object containing List of declarative workspace data filters.

`get_declarative_workspaces`() → *CatalogDeclarativeWorkspaces*

Get all workspaces in the current organization in a declarative form.

Args:
None

Returns:

CatalogDeclarativeWorkspaces:
Declarative Workspaces object including all the workspaces for given organization.

`get_workspace`(*workspace_id: str*) → *CatalogWorkspace*

Get an individual workspace.

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

Returns:

CatalogWorkspace:
Catalog workspace object containing structure of the workspace.

`list_workspaces`() → List[*CatalogWorkspace*]

Returns a list of all workspaces in current organization

Args:
List[CatalogWorkspace]

Returns:

List[CatalogWorkspace]:
List of workspaces in the current organization.

`load_and_put_declarative_workspace`(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

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

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`load_and_put_declarative_workspace_data_filters`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/goo sdk/checkouts/latest/gooddata-sdk/docs')*) → None

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.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`load_and_put_declarative_workspaces`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

This method combines load_declarative_workspaces and put_declarative_workspaces methods to load and set layouts stored using store_declarative_workspaces.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`load_declarative_workspace`(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeWorkspaceModel*

Load declarative workspaces layout, which was stored using store_declarative_workspace.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeWorkspaceModel:
Object Containing declarative Logical Data Model and declarative Analytical Model.

`load_declarative_workspace_data_filters`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeWorkspaceDataFilters*

Loads workspace data filters layout, which was stored using store_declarative_workspace_data_filters.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeWorkspaceDataFilters:
Object containing List of declarative workspace data filters.

`load_declarative_workspaces`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → *CatalogDeclarativeWorkspaces*

Load declarative workspaces layout, which was stored using store_declarative_workspaces

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:

CatalogDeclarativeWorkspaces:
Declarative Workspaces Object

`put_declarative_workspace`(*workspace_id: str*, *workspace:* CatalogDeclarativeWorkspaceModel) → None

Set a workspace layout.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

workspace (CatalogDeclarativeWorkspaceModel):
Object Containing declarative Logical Data Model and declarative Analytical Model.

Returns:
None

`put_declarative_workspace_data_filters`(*workspace_data_filters:* CatalogDeclarativeWorkspaceDataFilters) → None

Set workspace data filers layout.

Args:

workspace_data_filters (CatalogDeclarativeWorkspaceDataFilters):
Object containing List of declarative workspace data filters.

Returns:
None

`put_declarative_workspaces`(*workspace:* CatalogDeclarativeWorkspaces) → None

Set layout of all workspaces and their hierarchy. Parameter is in declarative form.

Args:

workspace (CatalogDeclarativeWorkspaces):
Declarative Workspaces object including all the workspaces for given organization.

Returns:
None

`store_declarative_workspace`(*workspace_id: str*, *layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store workspace layout in a directory hierarchy.

Args:

workspace_id (str):
Workspace identification string e.g. "demo"

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

`store_declarative_workspace_data_filters`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Store workspace data filters layout in a directory hierarchy.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

`store_declarative_workspaces`(*layout_root_path: Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/gooddata-sdk/checkouts/latest/gooddata-sdk/docs')*) → None

Stores declarative workspaces in a given path, as folder hierarchy.

Args:

layout_root_path (Path, optional):
Path to the root of the layout directory. Defaults to Path.cwd().

Returns:
None

3.1.2 gooddata_sdk.client

Module containing a class that provides access to metadata and afm services.

Classes

GoodDataApiClient(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

__init__(host, token[, custom_headers, ...])	Take url, token for connecting to GoodData.CN.

Attributes

actions_api
afm_client
custom_headers
entities_api
layout_api
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[, show_all_values])	

gooddata_sdk.compute.model.attribute.Attribute

class gooddata_sdk.compute.model.attribute.**Attribute**(*local_id: str*, *label: Union[ObjId, str]*, *show_all_values: Optional[bool] = None*)

Bases: *ExecModelEntity*

__init__(*local_id: str*, *label: Union[ObjId, str]*, *show_all_values: Optional[bool] = None*) → 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
- **show_all_values** – request show all values functionality for a given attribute

Methods

__init__(local_id, label[, show_all_values])	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
show_all_values

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

compute_model_to_api_model([attributes, ...])	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

BareExecutionResponse(api_client, ...)	Holds ExecutionResponse from triggered report computation and allows reading report's results.
Execution(api_client, workspace_id, ...)	An envelope class holding execution related classes:
ExecutionDefinition(attributes, metrics, ...)	
ExecutionResponse	alias of *Execution*
ExecutionResult(result)	
ResultCacheMetadata(result_cache_metadata)	
TotalDefinition(local_id, aggregation, ...)	
TotalDimension(idx[, items])	

gooddata_sdk.compute.model.execution.BareExecutionResponse

class gooddata_sdk.compute.model.execution.**BareExecutionResponse**(*api_client: GoodDataApiClient*, *workspace_id: str*, *execution_response: AfmExecutionResponse*)

Bases: `object`

Holds ExecutionResponse from triggered report computation and allows reading report's results.

__init__(*api_client: GoodDataApiClient*, *workspace_id: str*, *execution_response: AfmExecutionResponse*)

Methods

__init__(api_client, workspace_id, ...)	
read_result(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**(*api_client:* GoodDataApiClient, *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__(*api_client:* GoodDataApiClient, *workspace_id: str*, *exec_def:* ExecutionDefinition, *response: AfmExecutionResponse*)

Methods

__init__(api_client, workspace_id, exec_def, ...)	
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_dimensions_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.ResultCacheMetadata

class gooddata_sdk.compute.model.execution.**ResultCacheMetadata**(*result_cache_metadata: ResultCacheMetadata*)

Bases: object

__init__(*result_cache_metadata: ResultCacheMetadata*)

Methods

__init__(result_cache_metadata)
check_bytes_size_limit([result_size_bytes_limit])

Attributes

afm
execution_response
result_size
result_spec

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

__init__(local_id, aggregation, ...)	Method generated by attrs for class TotalDefinition.

Attributes

local_id	total's local identifier
aggregation	aggregation function; case insensitive; one of SUM, MIN, MAX, MED, AVG
metric_local_id	local identifier of the measure to calculate total for
total_dims	

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

__init__(idx[, items])	Method generated by attrs for class TotalDimension.

Attributes

idx	index of dimension in which to calculate the total
items	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

ResultSizeBytesLimitExceeded(...)	
ResultSizeDimensionsLimitsExceeded(...)	

gooddata_sdk.compute.model.execution.ResultSizeBytesLimitExceeded

exception gooddata_sdk.compute.model.execution.**ResultSizeBytesLimitExceeded**(*result_size_bytes_limit: int*, *actual_result_bytes_size: int*)

gooddata_sdk.compute.model.execution.ResultSizeDimensionsLimitsExceeded

exception gooddata_sdk.compute.model.execution.**ResultSizeDimensionsLimitsExceeded**(*result_size_dimensions_limi*
Tuple[Optional[int], ...], *actual_result_size_dimensions*
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: Optional[list[str]] = None*)

Bases: *Filter*

__init__(*label: Union[ObjId, str, Attribute]*, *values: Optional[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: Optional[list[str]] = None*)

Bases: *AttributeFilter*

__init__(*label: Union[ObjId, str, Attribute]*, *values: Optional[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: Optional[list[str]] = None*)

Bases: *AttributeFilter*

__init__(*label: Union[ObjId, str, Attribute]*, *values: Optional[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.PopDatesetMetric

class gooddata_sdk.compute.model.metric.**PopDatesetMetric**(*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: Optional[list[*Filter*]] = None*)

Bases: *Metric*

__init__(*local_id: str*, *item:* ObjId, *aggregation: Optional[str] = None*, *compute_ratio: bool = False*, *filters: Optional[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

ComputeService(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

__init__(api_client)	
for_exec_def(workspace_id, exec_def)	Starts computation in GoodData.CN workspace, using the provided execution definition.
retrieve_result_cache_metadata(workspace_id, ...)	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

retrieve_result_cache_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

Insight(from_vis_obj[, side_loads])	
InsightAttribute(attribute)	
InsightBucket(bucket)	
InsightFilter(f)	
InsightMetric(metric)	Represents metric placed on an insight.
InsightService(api_client)	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
show_all_values

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

__init__(api_client)	
get_insight(workspace_id, insight_id)	Gets a single insight from a workspace.
get_insights(workspace_id)	Gets all insights for a workspace.

get_insight(*workspace_id: str*, *insight_id: str*) → *Insight*

Gets a single insight from a workspace.

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

insight_id (str):
Insight identifier string e.g. “bikes”

Returns:

Insight:
A single Insight object contains side loaded metadata about the entities it references

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.

Args:

workspace_id (str):
Workspace identification string e.g. “demo”

Returns:

list[Insight]:
All available insights, each insight will contain side loaded metadata about the entities it references

3.1.5 gooddata_sdk.sdk

Classes

GoodDataSdk(client)	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

__init__(client)	Take instance of GoodDataApiClient and return new GoodDataSdk instance.
create(host_, token_[, extra_user_agent_])	Create common GoodDataApiClient and return new GoodDataSdk instance.
create_from_profile([profile, profiles_path])	

Attributes

catalog_data_source
catalog_organization
catalog_permission
catalog_user
catalog_workspace
catalog_workspace_content
client
compute
insights
support
tables

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

ExecutionTable(response, first_page)	Represents execution result as a table.
TableService(api_client)	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

__init__(response, first_page)	
read_all()	Returns a generator that will be yielding execution result as rows.

Attributes

attributes	
column_ids	Returns column identifiers.
column_metadata	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

build_stores()	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

AttributeConverterStore()	Store for conversion of attributes
Converter()	Base Converter class.
ConverterRegistryStore()	Class store TypeConverterRegistry instances for each registered type.
DBTypeConverterStore()	Store for conversion of database types
DateConverter()	
DatetimeConverter()	
IntegerConverter()	
StringConverter()	
TypeConverterRegistry(type_name)	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

__init__()	

find_converter(type_name[, sub_type])	Find Converter for given type and sub type.
register(type_name, class_converter[, sub_types])	Register Converter instance created from provided Converter class to given type and list of sub types.
reset()	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__()	

find_converter(type_name[, sub_type])	Find Converter for given type and sub type.
register(type_name, class_converter[, sub_types])	Register Converter instance created from provided Converter class to given type and list of sub types.
reset()	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.**DBTypeConverterStore**

Bases: *ConverterRegistryStore*

Store for conversion of database types

__init__()

Methods

__init__()	

find_converter(type_name[, sub_type])	Find Converter for given type and sub type.
register(type_name, class_converter[, sub_types])	Register Converter instance created from provided Converter class to given type and list of sub types.
reset()	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.**DateConverter**

Bases: *Converter*

__init__()

Methods

__init__()	
db_data_type()	
set_external_fnc(fnc)	
to_date(value)	Add first month and first date to incomplete iso date string.
to_external_type(value)	
to_type(value)	

Attributes

DEFAULT_DB_DATA_TYPE

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

__init__()	
db_data_type()	
set_external_fnc(fnc)	
to_datetime(value)	Append minutes to incomplete datetime string.
to_external_type(value)	
to_type(value)	

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.**StringConverter**

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.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)	
good_pandas_profile_content([profile, ...])	This is workaround for GoodPandas.
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[, ...])	
mandatory_profile_content_check(profile, ...)	
profile_content([profile, profiles_path])	
read_layout_from_file(path)	
recreate_directory(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.good_pandas_profile_content

gooddata_sdk.utils.**good_pandas_profile_content**(*profile: str = 'default'*, *profiles_path: Path = PosixPath('/home/docs/.gooddata/profiles.yaml')*) → Tuple[Dict[str, Any], Dict[str, Any]]

This is workaround for GoodPandas. We should only use profile_content in the future. For that we need to unify GoodPandas and GoodDataSdk interface.

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_api_client as api_client
>>> import gooddata_api_client.apis as apis
>>> api = apis.EntitiesApi(api_client.ApiClient())
>>> 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.mandatory_profile_content_check

gooddata_sdk.utils.**mandatory_profile_content_check**(*profile: str*, *profile_content_keys: KeysView*) → None

gooddata_sdk.utils.profile_content

gooddata_sdk.utils.**profile_content**(*profile: str = 'default'*, *profiles_path: Path = PosixPath('/home/docs/.gooddata/profiles.yaml')*) → dict[str, Any]

gooddata_sdk.utils.read_layout_from_file

gooddata_sdk.utils.**read_layout_from_file**(*path: Path*) → Any

gooddata_sdk.utils.recreate_directory

gooddata_sdk.utils.**recreate_directory**(*path: Path*) → None

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

__init__()	
count(value, /)	Return number of occurrences of value.
index(value[, start, stop])	Return first index of value.

Attributes

data	Alias for field number 0
included	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

__init__(stream[, default_style, ...])
add_implicit_resolver(tag, regexp, first)
add_multi_representer(data_type, representer)
add_path_resolver(tag, path[, kind])
add_representer(data_type, representer)
analyze_scalar(scalar)
anchor_node(node)
ascend_resolver()
check_empty_document()
check_empty_mapping()
check_empty_sequence()
check_resolver_prefix(depth, path, kind, ...)
check_simple_key()
choose_scalar_style()
close()
descend_resolver(current_node, current_index)
determine_block_hints(text)
dispose()

continues on next page

Table 1 – continued from previous page

emit(event)
expect_alias()
expect_block_mapping()
expect_block_mapping_key([first])
expect_block_mapping_simple_value()
expect_block_mapping_value()
expect_block_sequence()
expect_block_sequence_item([first])
expect_document_end()
expect_document_root()
expect_document_start([first])
expect_first_block_mapping_key()
expect_first_block_sequence_item()
expect_first_document_start()
expect_first_flow_mapping_key()
expect_first_flow_sequence_item()
expect_flow_mapping()
expect_flow_mapping_key()
expect_flow_mapping_simple_value()
expect_flow_mapping_value()
expect_flow_sequence()
expect_flow_sequence_item()
expect_node([root, sequence, mapping, ...])
expect_nothing()
expect_scalar()

continues on next page

Table 1 – continued from previous page

expect_stream_start()
flush_stream()
generate_anchor(node)
ignore_aliases(data)
increase_indent([flow, indentless])
need_events(count)
need_more_events()
open()
prepare_anchor(anchor)
prepare_tag(tag)
prepare_tag_handle(handle)
prepare_tag_prefix(prefix)
prepare_version(version)
process_anchor(indicator)
process_scalar()
process_tag()
represent(data)
represent_binary(data)
represent_bool(data)
represent_data(data)
represent_date(data)
represent_datetime(data)
represent_dict(data)
represent_float(data)
represent_int(data)

continues on next page

Table 1 – continued from previous page

represent_list(data)
represent_mapping(tag, mapping[, flow_style])
represent_none(data)
represent_scalar(tag, value[, style])
represent_sequence(tag, sequence[, flow_style])
represent_set(data)
represent_str(data)
represent_undefined(data)
represent_yaml_object(tag, data, cls[, ...])
resolve(kind, value, implicit)
serialize(node)
serialize_node(node, parent, index)
write_double_quoted(text[, split])
write_folded(text)
write_indent()
write_indicator(indicator, need_whitespace)
write_line_break([data])
write_literal(text)
write_plain(text[, split])
write_single_quoted(text[, split])
write_stream_end()
write_stream_start()
write_tag_directive(handle_text, prefix_text)
write_version_directive(version_text)

Attributes

ANCHOR_TEMPLATE
DEFAULT_MAPPING_TAG
DEFAULT_SCALAR_TAG
DEFAULT_SEQUENCE_TAG
DEFAULT_TAG_PREFIXES
ESCAPE_REPLACEMENTS
inf_value
yaml_implicit_resolvers
yaml_multi_representers
yaml_path_resolvers
yaml_representers

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

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W