GoodData SDK

Release 0.7.0

GoodData Corporation

Jul 14, 2022

CONTENTS:

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

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

CHAPTER

ONE

GETTING STARTED

1.1 Requirements

- GoodData.CN installation; either running on your cloud infrastructure or the free Community Edition running on your workstation
- Python 3.7 or newer

1.2 Installation

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

```
pip install gooddata-sdk
```

CHAPTER

TWO

SERVICES

All the services are accessible by class `gooddata_sdk.GoodDataSdk`. The class forms an entry-point to the SDK. All other examples assume you have an entry-point to the GoodDataSdk instance already initialized.

Example of how to create an instance of GoodDataSdk:

```
import gooddata_sdk

# 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 = gooddata_sdk.GoodDataSdk.create(host, token)
```

2.1 Catalog Service

The `gooddata_sdk.catalog` service enables you to list catalog objects from a workspace. It contains all the datasets and metrics registered in the workspace.

Example of how to read all datasets and metrics in a workspace:

```
workspace_id = "demo"

# read catalog for demo workspace
catalog = sdk.catalog.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))
```

2.2 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*

Example of how to read all insights in a workspace:

```
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))
```

2.3 Compute Service

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

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

2.4 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.

For example look at how you can get tabular data for an insight defined on your GoodData.CN server:

```
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)

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

CHAPTER

THREE

API DOCUMENTATION

Check out the *API* section for further information.

3.1 API

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

3.1.1 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.compute_model	
gooddata_sdk.insight	
gooddata_sdk.sdk	
gooddata_sdk.table	
gooddata_sdk.type_converter	
gooddata_sdk.utils	

gooddata_sdk.catalog

Classes

Catalog(valid_obj_fun, datasets, metrics)	
CatalogAttribute(attribute, labels)	
CatalogDataset(dataset, attributes, facts)	
CatalogEntry()	
CatalogFact(fact)	
CatalogLabel(label)	
CatalogMetric(metric)	
CatalogService(api_client)	

gooddata_sdk.catalog.Catalog

class gooddata_sdk.catalog.**Catalog**(*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.
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[*gooddata_sdk.compute_model.Attribute, gooddata_sdk.compute_model.Metric, gooddata_sdk.compute_model.Filter, gooddata_sdk.catalog.CatalogLabel, gooddata_sdk.catalog.CatalogFact, gooddata_sdk.catalog.CatalogMetric, *List[Union[*gooddata_sdk.compute_model.Attribute, gooddata_sdk.compute_model.Metric, gooddata_sdk.compute_model.Filter, gooddata_sdk.catalog.CatalogLabel, gooddata_sdk.catalog.CatalogFact, gooddata_sdk.catalog.CatalogMetric*]],* gooddata_sdk.compute.ExecutionDefinition*]*) → *gooddata_sdk.catalog.Catalog*

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

Returns

find_label_attribute(*id_obj: Union[str,* gooddata_sdk.compute_model.ObjId*, Dict[str, Dict[str, str]], Dict[str, str]]*) → Optional[*gooddata_sdk.catalog.CatalogAttribute*]

Get attribute by label id.

get_dataset(*dataset_id: Union[str,* gooddata_sdk.compute_model.ObjId*]*) → Optional[*gooddata_sdk.catalog.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

:rtype CatalogDataset

get_metric(*metric_id: Union[str,* gooddata_sdk.compute_model.ObjId*]*) → Optional[*gooddata_sdk.catalog.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

:rtype CatalogMetric

gooddata_sdk.catalog.CatalogAttribute

class gooddata_sdk.catalog.**CatalogAttribute**(*attribute: dict[str, Any]*, *labels: list[*CatalogLabel*]*)

Bases: *gooddata_sdk.catalog.CatalogEntry*

__init__(*attribute: dict[str, Any]*, *labels: list[*CatalogLabel*]*) → None

Methods

__init__(attribute, labels)
as_computable()
find_label(id_obj)
primary_label()

Attributes

dataset
description
granularity
id
labels
obj_id
title
type

gooddata_sdk.catalog.CatalogDataset

class gooddata_sdk.catalog.**CatalogDataset**(*dataset: dict[str, Any]*, *attributes: list[*CatalogAttribute*]*, *facts: list[*CatalogFact*]*)

Bases: *gooddata_sdk.catalog.CatalogEntry*

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

Methods

__init__(dataset, 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[*gooddata_sdk.catalog.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.CatalogEntry

class gooddata_sdk.catalog.**CatalogEntry**
Bases: object

__init__()

Methods

__init__()

Attributes

description
id
obj_id
title
type

gooddata_sdk.catalog.CatalogFact

class gooddata_sdk.catalog.**CatalogFact**(*fact: dict[str, Any]*)
Bases: *gooddata_sdk.catalog.CatalogEntry*

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

Methods

__init__(fact)
as_computable()

Attributes

description
id
obj_id
title
type

gooddata_sdk.catalog.CatalogLabel

class gooddata_sdk.catalog.**CatalogLabel**(*label: dict[str, Any]*)

Bases: *gooddata_sdk.catalog.CatalogEntry*

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

Methods

__init__(label)
as_computable()

Attributes

description
id
obj_id
primary
title
type

gooddata_sdk.catalog.CatalogMetric

class gooddata_sdk.catalog.**CatalogMetric**(*metric: dict[str, Any]*)
Bases: *gooddata_sdk.catalog.CatalogEntry*

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

Methods

__init__(metric)
as_computable()

Attributes

description
format
id
obj_id
title
type

gooddata_sdk.catalog.CatalogService

class gooddata_sdk.catalog.**CatalogService**(*api_client:* gooddata_sdk.client.GoodDataApiClient)
Bases: object

__init__(*api_client:* gooddata_sdk.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_full_catalog(workspace_id)	Retrieves catalog for a workspace.

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

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

Parameters

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

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

get_full_catalog(*workspace_id: str*) → *gooddata_sdk.catalog.Catalog*

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

Parameters **workspace_id** – workspace identifier

Returns

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

afm_client
metadata_client
scan_client

gooddata_sdk.compute

Classes

ComputeService(api_client)	Compute service drives computation of analytics for a GoodData.CN workspaces.
ExecutionDefinition(attributes, metrics, ...)	
ExecutionResponse(actions_api, workspace_id, ...)	
ExecutionResult(result)	

gooddata_sdk.compute.ComputeService

class gooddata_sdk.compute.**ComputeService**(*api_client:* gooddata_sdk.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:* gooddata_sdk.client.GoodDataApiClient)

Methods

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

for_exec_def(*workspace_id: str*, *exec_def:* gooddata_sdk.compute.ExecutionDefinition) → *gooddata_sdk.compute.ExecutionResponse*

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

Returns

gooddata_sdk.compute.ExecutionDefinition

class gooddata_sdk.compute.**ExecutionDefinition**(*attributes: Optional[list[Attribute]]*, *metrics: Optional[list[Metric]]*, *filters: Optional[list[Filter]]*, *dimensions: list[Optional[list[str]]]*)

Bases: `object`

__init__(*attributes: Optional[list[Attribute]]*, *metrics: Optional[list[Metric]]*, *filters: Optional[list[Filter]]*, *dimensions: list[Optional[list[str]]]*) → 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.ExecutionResponse

class gooddata_sdk.compute.**ExecutionResponse**(*actions_api: gooddata_afm_client.api.actions_api.ActionsApi*, *workspace_id: str*, *exec_def:* gooddata_sdk.compute.ExecutionDefinition, *response: gooddata_afm_client.model.afm_execution_response.AfmExecutionResponse*)

Bases: object

__init__(*actions_api: gooddata_afm_client.api.actions_api.ActionsApi*, *workspace_id: str*, *exec_def:* gooddata_sdk.compute.ExecutionDefinition, *response: gooddata_afm_client.model.afm_execution_response.AfmExecutionResponse*)

Methods

__init__(actions_api, workspace_id, ...)	
read_result(limit[, offset])	Reads from the execution result.

Attributes

exec_def
result_id
workspace_id

read_result(*limit: Union[int, list[int]]*, *offset: Union[None, int, list[int]] = None*) → *ExecutionResult*

Reads from the execution result. :param offset: :param limit: :return:

gooddata_sdk.compute.ExecutionResult

class gooddata_sdk.compute.**ExecutionResult**(*result: gooddata_afm_client.model.execution_result.ExecutionResult*)

Bases: object

__init__(*result: gooddata_afm_client.model.execution_result.ExecutionResult*)

Methods

__init__(result)
get_all_header_values(dim, header_idx)
is_complete([dim])
next_page_start([dim])

Attributes

data
grand_totals
headers
paging
paging_count
paging_offset
paging_total

gooddata_sdk.compute_model

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.compute_model_to_api_model

gooddata_sdk.compute_model.**compute_model_to_api_model**(*attributes: Optional[list[Attribute]] = None*, *metrics: Optional[list[Metric]] = None*, *filters: Optional[list[Filter]] = None*) → afm_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

Returns

Classes

AbsoluteDateFilter(dataset, from_date, to_date)	
AllTimeFilter()	Filter that is semantically equivalent to absent filter.
ArithmeticMetric(local_id, operator, operands)	
Attribute(local_id, label)	
AttributeFilter(label[, values])	
ExecModelEntity()	
Filter()	
Metric(local_id)	
MetricValueFilter(metric, operator, values)	
NegativeAttributeFilter(label[, values])	
ObjId(id, type)	
PopDate(attribute, periods_ago)	
PopDateDataset(dataset, periods_ago)	
PopDateMetric(local_id, metric, date_attributes)	
PopDatesetMetric(local_id, metric, date_datasets)	
PositiveAttributeFilter(label[, values])	

continues on next page

Table 31 – continued from previous page

RankingFilter(metrics, operator, value, ...)	
RelativeDateFilter(dataset, granularity, ...)	
SimpleMetric(local_id, item[, aggregation, ...])	

gooddata_sdk.compute_model.AbsoluteDateFilter

class gooddata_sdk.compute_model.**AbsoluteDateFilter**(*dataset:* gooddata_sdk.compute_model.ObjId, *from_date: str*, *to_date: str*)

Bases: *gooddata_sdk.compute_model.Filter*

__init__(*dataset:* gooddata_sdk.compute_model.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.AllTimeFilter

class gooddata_sdk.compute_model.**AllTimeFilter**

Bases: *gooddata_sdk.compute_model.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.ArithmeticMetric

class gooddata_sdk.compute_model.**ArithmeticMetric**(*local_id: str*, *operator: str*, *operands: list[Union[str,* Metric*]]*)

Bases: *gooddata_sdk.compute_model.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.Attribute

class gooddata_sdk.compute_model.**Attribute**(*local_id: str*, *label: Union[gooddata_sdk.compute_model.ObjId, str]*)

Bases: *gooddata_sdk.compute_model.ExecModelEntity*

__init__(*local_id: str*, *label: Union[gooddata_sdk.compute_model.ObjId, str]*) → None

Creates new attribute that can be used to slice or dice metric values during computation.

Parameters

- **local_id** – identifier of the attribute within the execution
- **label** – identifier of the label to use for slicing or dicing; specified either as ObjId or str containing the label id

Methods

__init__(local_id, label)	Creates new attribute that can be used to slice or dice metric values during computation.
as_api_model()	
has_same_label(other)	

Attributes

label
local_id

gooddata_sdk.compute_model.AttributeFilter

class gooddata_sdk.compute_model.**AttributeFilter**(*label: Union[ObjId, str, Attribute]*, *values: list[str] = None*)

Bases: *gooddata_sdk.compute_model.Filter*

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

Methods

__init__(label[, values])
as_api_model()
is_noop()

Attributes

apply_on_result
label
values

gooddata_sdk.compute_model.ExecModelEntity

class gooddata_sdk.compute_model.**ExecModelEntity**

Bases: object

__init__() → None

Methods

__init__()
as_api_model()

gooddata_sdk.compute_model.Filter

class gooddata_sdk.compute_model.**Filter**

Bases: *gooddata_sdk.compute_model.ExecModelEntity*

__init__() → None

Methods

__init__()
as_api_model()
is_noop()

Attributes

apply_on_result

gooddata_sdk.compute_model.Metric

class gooddata_sdk.compute_model.**Metric**(*local_id: str*)

Bases: *gooddata_sdk.compute_model.ExecModelEntity*

__init__(*local_id: str*) → None

Methods

__init__(local_id)
as_api_model()

Attributes

local_id

gooddata_sdk.compute_model.MetricValueFilter

class gooddata_sdk.compute_model.**MetricValueFilter**(*metric: Union[ObjId, str, Metric]*, *operator: str*, *values: Union[float, int, tuple[float, float]]*, *treat_nulls_as: Union[float, None] = None*)

Bases: *gooddata_sdk.compute_model.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.NegativeAttributeFilter

class gooddata_sdk.compute_model.**NegativeAttributeFilter**(*label: Union[ObjId, str, Attribute]*, *values: list[str] = None*)

Bases: *gooddata_sdk.compute_model.AttributeFilter*

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

Methods

__init__(label[, values])
as_api_model()
is_noop()

Attributes

apply_on_result
label
values

gooddata_sdk.compute_model.ObjId

class gooddata_sdk.compute_model.**ObjId**(*id: str*, *type: str*)

Bases: object

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

Methods

__init__(id, type)
as_afm_id()
as_identifier()

Attributes

id
type

gooddata_sdk.compute_model.PopDate

class gooddata_sdk.compute_model.**PopDate**(*attribute: Union[*gooddata_sdk.compute_model.ObjId, gooddata_sdk.compute_model.Attribute*]*, *periods_ago: int*)

Bases: object

__init__(*attribute: Union[*gooddata_sdk.compute_model.ObjId, gooddata_sdk.compute_model.Attribute*]*, *periods_ago: int*) → None

Methods

__init__(attribute, periods_ago)
as_api_model()

Attributes

attribute
periods_ago

gooddata_sdk.compute_model.PopDateDataset

class gooddata_sdk.compute_model.**PopDateDataset**(*dataset: Union[*gooddata_sdk.compute_model.ObjId*, str]*, *periods_ago: int*)

Bases: object

__init__(*dataset: Union[*gooddata_sdk.compute_model.ObjId*, str]*, *periods_ago: int*) → None

Methods

__init__(dataset, periods_ago)
as_api_model()

Attributes

dataset
periods_ago

gooddata_sdk.compute_model.PopDateMetric

class gooddata_sdk.compute_model.**PopDateMetric**(*local_id: str*, *metric: Union[str,* Metric*]*, *date_attributes: list[*PopDate*]*)

Bases: *gooddata_sdk.compute_model.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.PopDatesetMetric

class gooddata_sdk.compute_model.**PopDatesetMetric**(*local_id: str*, *metric: Union[str,* Metric*]*, *date_datasets: list[*PopDateDataset*]*)

Bases: *gooddata_sdk.compute_model.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.PositiveAttributeFilter

class gooddata_sdk.compute_model.**PositiveAttributeFilter**(*label: Union[*ObjId*, str,* Attribute*]*, *values: list[str] = None*)

Bases: *gooddata_sdk.compute_model.AttributeFilter*

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

Methods

__init__(label[, values])
as_api_model()
is_noop()

Attributes

apply_on_result
label
values

gooddata_sdk.compute_model.RankingFilter

class gooddata_sdk.compute_model.**RankingFilter**(*metrics: list[Union[ObjId, Metric, str]]*, *operator: str*, *value: int*, *dimensionality: Optional[list[Union[str, ObjId, Attribute, Metric]]]*)

Bases: *gooddata_sdk.compute_model.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

continues on next page

Table 64 – continued from previous page

operator
value

gooddata_sdk.compute_model.RelativeDateFilter

class gooddata_sdk.compute_model.**RelativeDateFilter**(*dataset:* gooddata_sdk.compute_model.ObjId, *granularity: str*, *from_shift: int*, *to_shift: int*)

Bases: *gooddata_sdk.compute_model.Filter*

__init__(*dataset:* gooddata_sdk.compute_model.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.SimpleMetric

class gooddata_sdk.compute_model.**SimpleMetric**(*local_id: str*, *item:* ObjId, *aggregation: Optional[str] = None*, *compute_ratio: bool = False*, *filters: list[*Filter*] = None*)

Bases: *gooddata_sdk.compute_model.Metric*

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

Methods

__init__(local_id, item[, aggregation, ...])
as_api_model()

Attributes

aggregation
compute_ratio
filters
item
local_id

gooddata_sdk.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

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.

continues on next page

Table 78 – continued from previous page

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. :return: local_id of master metric, None if not a time comparison metric

gooddata_sdk.insight.InsightService

class gooddata_sdk.insight.**InsightService**(*api_client:* gooddata_sdk.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:* gooddata_sdk.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*) → *gooddata_sdk.insight.Insight*
Gets a single insight from a workspace.

Parameters

- **workspace_id** – identifier of workspace to load insight from
- **insight_id** – identifier of the insight

Returns single insight; the insight will contain sideloaded metadata about the entities it references

Return type *Insight*

get_insights(*workspace_id: str*) → list[*Insight*]
Gets all insights for a workspace. The insights will contain side loaded metadata for all execution entities that they reference.

Parameters **workspace_id** – identifier of workspace to load insights from

Returns all available insights, each insight will contain side loaded metadata about the entities it references

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:* gooddata_sdk.client.GoodDataApiClient)

Bases: `object`

Top-level class that wraps all the functionality together.

__init__(*client:* gooddata_sdk.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.

Attributes

`catalog`
`compute`
`insights`
`tables`

classmethod create(*host_: str*, *token_: str*, *extra_user_agent_: Optional[str] = None*, ***custom_headers_: Optional[str]*) → *gooddata_sdk.sdk.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.

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:* gooddata_sdk.compute.ExecutionResponse, *first_page:* gooddata_sdk.compute.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:* gooddata_sdk.compute.ExecutionResponse, *first_page:* gooddata_sdk.compute.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.

continues on next page

Table 85 – continued from previous page

metrics

property column_ids: list[str]
Returns column identifiers. Each row will be a mapping of column identifier to column data.

Returns

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. :return:

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:* gooddata_sdk.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:* gooddata_sdk.client.GoodDataApiClient) → None

Methods

__init__(api_client)
for_insight(workspace_id, insight)
for_items(workspace_id, items[, filters])

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: *gooddata_sdk.type_converter.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_[, 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*) → *gooddata_sdk.type_converter.Converter*
Find Converter for given type and sub type. :param type_name: type name :param sub_type: sub type name

classmethod register(*type_name: str*, *class_: 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. :param type_name: type

name :param class_: Converter class :param 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_[, 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*) → *gooddata_sdk.type_converter.Converter*

Find Converter for given type and sub type. :param type_name: type name :param sub_type: sub type name

classmethod register(*type_name: str*, *class_: 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. :param type_name: type name :param class_: Converter class :param 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: *gooddata_sdk.type_converter.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_[, 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*) → *gooddata_sdk.type_converter.Converter*

Find Converter for given type and sub type. :param type_name: type name :param sub_type: sub type name

classmethod register(*type_name: str*, *class_: 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. :param type_name: type name :param class_: Converter class :param 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: *gooddata_sdk.type_converter.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*) → datetime.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: *gooddata_sdk.type_converter.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.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: *gooddata_sdk.type_converter.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: *gooddata_sdk.type_converter.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 :param type_name: type name

Methods

__init__(type_name)	Initialize instance with type for which instance is going to be responsible :param type_name: type name
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]*) → *gooddata_sdk.type_converter.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. :param sub_type: sub-type name :return: Converter instance

register(*converter:* gooddata_sdk.type_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. :param converter: converter instance :param sub_type: sub-type name

gooddata_sdk.utils

Functions

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.

gooddata_sdk.utils.id_obj_to_key

gooddata_sdk.utils.**id_obj_to_key**(*id_obj: Union[str,* gooddata_sdk.compute_model.ObjId*, Dict[str, Dict[str, str]], Dict[str, str]]*) → str

Given an object containing an id+type pair, this function will return a string key.

For convenience, this also recognizes the *ref* format used by GoodData.UI SDK. In that format, the id+type are wrapped in 'identifier'.

Parameters **id_obj** – id object

Returns string that can be used as key

gooddata_sdk.utils.load_all_entities

gooddata_sdk.utils.**load_all_entities**(*get_page_func: functools.partial[Any]*, *page_size: int = 500*) → *AllPagedEntities*

Loads all entities from a paged resource. The primary input to this function is a partial function that is setup with all the fixed parameters. Given this the function will get entities page-by-page and merge them into a single 'pseudo-response' containing data and included attributes.

An example usage:

```
>>> import functools
>>> import gooddata_metadata_client as metadata_client
>>> import gooddata_metadata_client.apis as metadata_apis
>>> api = metadata_apis.EntitiesApi(metadata_client.ApiClient())
>>> 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

Returns

Classes

AllPagedEntities(data, included)	
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.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)

3.2 Indices and Tables

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T