TM1py Documentation

Release 2.0

Marius Wirtz

Jan 30, 2024

CONTENTS

By wrapping the IBM Planning Analytics (TM1) REST API in a concise Python framework, TM1py facilitates Python developments for TM1.

Interacting with TM1 programmatically has never been easier.

```
with TM1Service(address='localhost', port=8001, user='admin', password='apple',␣
↪ssl=True) as tm1:
    subset = Subset(dimension_name='Month', subset_name='Q1', elements=['Jan', 'Feb',
↪'Mar'])
    tm1.subsets.create(subset, private=True)
```

TM1py offers handy features to interact with TM1 from Python, such as

- Read data from cubes through cube views and MDX Queries
- Write data into cubes
- Execute processes and chores
- Execute loose statements of TI
- CRUD features for TM1 objects (cubes, dimensions, subsets, etc.)
- Query and kill threads
- Query MessageLog, TransactionLog and AuditLog
- Generate MDX Queries from existing cube views

CHAPTER

ONE

REQUIREMENTS

- python (3.7 or higher)
- requests
- requests_negotiate_sspi
- TM1 11

CHAPTER

TWO

OPTIONAL REQUIREMENTS

- pandas

CHAPTER

THREE

INSTALL

without pandas

```
pip install tm1py
```

with pandas

```
pip install "tm1py[pandas]"
```

CHAPTER

FOUR

USAGE

on-premise

```
from TM1py.Services import TM1Service

with TM1Service(address='localhost', port=8001, user='admin', password='apple',␣
↪ssl=True) as tm1:
    for chore in tm1.chores.get_all():
        chore.reschedule(hours=-1)
        tm1.chores.update(chore)
```

IBM cloud

```
with TM1Service(
        base_url='https://mycompany.planning-analytics.ibmcloud.com/tm1/api/tm1/',
        user="non_interactive_user",
        namespace="LDAP",
        password="U3lSn5QLwoQZY2",
        ssl=True,
        verify=True,
        async_requests_mode=True) as tm1:
    for chore in tm1.chores.get_all():
        chore.reschedule(hours=-1)
        tm1.chores.update(chore)
```

4.1 API Documentation

If you are looking for information on a specific function, class, or method, this part of the documentation is for you.

4.1.1 Developer Interface

This part of the documentation covers all the classes of TM1py.

TM1 Services

class TM1py.**AnnotationService**(*rest:* RestService)

Service to handle Object Updates for TM1 CellAnnotations

create(*annotation:* Annotation, ***kwargs*) → Response

create an Annotation

Parameters
annotation – instance of TM1py.Annotation

create_many(*annotations: Iterable[*Annotation*]*, ***kwargs*) → Response

create an Annotation

Parameters
annotations – instances of TM1py.Annotation

delete(*annotation_id: str*, ***kwargs*) → Response

delete Annotation

Parameters
annotation_id – string, the id of the annotation

get(*annotation_id: str*, ***kwargs*) → *Annotation*

get an annotation from any cube through its unique id

Parameters
annotation_id – String, the id of the annotation

get_all(*cube_name: str*, ***kwargs*) → List[*Annotation*]

get all annotations from given cube as a List.

Parameters
cube_name –

update(*annotation:* Annotation, ***kwargs*) → Response

update Annotation. updateable attributes: commentValue

Parameters
annotation – instance of TM1py.Annotation

class TM1py.**ApplicationService**(*tm1_rest: <module 'TM1py.Services.RestService' from '/home/docs/checkouts/readthedocs.org/user_builds/tm1py/checkouts/latest/TM1py/Services/Re*

Service to Read and Write TM1 Applications

create(*application:* Application | *DocumentApplication*, *private: bool = False*, ***kwargs*) → Response

Create Planning Analytics application

Parameters

- **application** – instance of Application
- **private** – boolean

Returns

create_document_from_file(*path_to_file: str*, *application_path: str*, *application_name: str*, *private: bool = False*, ***kwargs*) → Response

Create DocumentApplication in TM1 from local file

Parameters

- **path_to_file** –
- **application_path** –
- **application_name** –
- **private** –

Returns

delete(*path: str*, *application_type: str | ApplicationTypes*, *application_name: str*, *private: bool = False*, ***kwargs*) → Response

Delete Planning Analytics application reference

Parameters

- **path** – path through folder structure to delete the applications entry. For instance: "Finance/Reports"
- **application_type** – type of the to be deleted application entry
- **application_name** – name of the to be deleted application entry
- **private** – Access level of the to be deleted object

Returns

exists(*path: str*, *application_type: str | ApplicationTypes*, *name: str*, *private: bool = False*, ***kwargs*) → bool

Check if application exists

Parameters

- **path** –
- **application_type** –
- **name** –
- **private** –

Returns

get(*path: str*, *application_type: str | ApplicationTypes*, *name: str*, *private: bool = False*, ***kwargs*) → *Application*

Retrieve Planning Analytics Application

Parameters

- **path** – path with forward slashes
- **application_type** – str or ApplicationType from Enum
- **name** –
- **private** –

Returns

get_all_private_root_names(***kwargs*)

get_all_public_root_names(***kwargs*)

get_document(*path: str*, *name: str*, *private: bool = False*, ***kwargs*) → DocumentApplication

Get Excel Application from TM1 Server in binary format. Can be dumped to file.

Parameters

- **path** – path through folder structure to application. For instance: "Finance/P&L.xlsx"
- **name** – name of the application
- **private** – boolean

Returns

Return DocumentApplication

rename(*path: str*, *application_type: str | ApplicationTypes*, *application_name: str*, *new_application_name: str*, *private: bool = False*, ***kwargs*)

update(*application:* Application *| DocumentApplication*, *private: bool = False*, ***kwargs*) → Response

Update Planning Analytics application

Parameters

- **application** – instance of Application
- **private** – boolean

Returns

update_document_from_file(*path_to_file: str*, *application_path: str*, *application_name: str*, *private: bool = False*, ***kwargs*) → Response

Update DocumentApplication in TM1 from local file

Parameters

- **path_to_file** –
- **application_path** –
- **application_name** –
- **private** –

Returns

update_or_create_document_from_file(*path: str*, *name: str*, *path_to_file: str*, *private: bool = False*, ***kwargs*) → Response

Update or create application from file

Parameters

- **path** – application path on server, i.e. 'Finance/Reports'
- **name** – name of the application on server, i.e. 'Flash.xlsx'
- **path_to_file** – full local file path of file, i.e. 'C:UsersUserFlash.xslx'
- **private** – access level of the object

Returns

Response

class TM1py.**CellService**(*tm1_rest:* RestService)

Service to handle Read and Write operations to TM1 cubes

activate_transactionlog(**args: str*, ***kwargs*) → Response

Activate Transactionlog for one or many cubes

Parameters
args – one or many cube names

Returns

begin_changeset() → str

begin a change set

Returns
Change set ID

check_cell_feeders(*cube_name: str*, *elements: Iterable | str*, *dimensions: Iterable[str] = None*, *sandbox_name: str = None*, *element_separator: str = ','*, *hierarchy_separator: str = '&&'*, *hierarchy_element_separator: str = '::'*, ***kwargs*) → Dict

Check feeders

Parameters

- **cube_name** – name of the target cube
- **elements** –

string "Hierarchy1::Element1 && Hierarchy2::Element4, Element9, Element2"

- Dimensions are not specified! They are derived from the position.
- The , separates the element-selections
- If more than one hierarchy is selected per dimension && splits the elementselections
- If no Hierarchy is specified. Default Hierarchy will be addressed

or Iterable [Element1, Element2, Element3] :param dimensions: optional. Dimension names in their natural order. Will speed up the execution! :param sandbox_name: str :param element_separator: Alternative separator for the elements, if elements are passed as string :param hierarchy_separator: Alternative separator for multiple hierarchies, if elements are passed as string :param hierarchy_element_separator: Alternative separator between hierarchy name and element name, if elements are passed as string :return: fed cell descriptor

clear(*cube: str*, ***kwargs*)

Takes the cube name and keyword argument pairs of dimensions and MDX expressions:

``` tm1.cells.clear(

cube="Sales", salesregion="{[Sales Region].[Australia],[Sales Region].[New Zealand]}", product="{[Product].[ABC]}", time="{[Time].[2022].Children}")

```

Make sure that the keyword argument names (e.g. product) map with the dimension names (e.g. Product) in the cube. Spaces in the dimension name (e.g., "Sales Region") must be omitted in the keyword (e.g. "salesregion")

Parameters

- **cube** – name of the cube
- **kwargs** – keyword argument pairs of dimension names and mdx set expressions

Returns

clear_spread(*cube: str*, *unique_element_names: Iterable[str]*, *sandbox_name: str = None*, ***kwargs*) → Response

Execute clear spread :param cube: name of the cube :param unique_element_names: target cell coordinates as unique element names (e.g. ["[d1].[c1]","[d2].[e3]"]) :param sandbox_name: str :return:

clear_with_dataframe(*cube: str*, *df: DataFrame*, *dimension_mapping: Dict = None*, ***kwargs*)

Clears data from a TM1 cube based on the distinct values in a DataFrame over cube dimensions.

Note:

This function is similar to *tm1.cells.clear*, but it is designed specifically for clearing data based on distinct values in a DataFrame over cube dimensions. The key difference is that this function interprets the DataFrame columns as dimensions and supports a mapping (*dimension_mapping*) for specifying hierarchies within those dimensions.

Parameters

- **cube** – str The name of the TM1 cube.
- **df** – pd.DataFrame The DataFrame containing distinct values over cube dimensions. Columns in the DataFrame should correspond to cube dimensions.
- **dimension_mapping** – Dict, optional A dictionary mapping the DataFrame columns to one or many hierarchies within the given dimension. If not provided, assumes that the dimensions have just one hierarchy.

Returns

None The function clears data in the specified TM1 cube.

Raises

ValueError – If there are unmatched dimensions in the DataFrame or if specified dimensions do not exist in the TM1 cube.

Example

```python
# Sample DataFrame with distinct values over cube dimensions data = {
    "Year": ["2021", "2022"], "Organisation": ["some_company", "some_company"],
    "Location": ["Germany", "Albania"]
}
# Sample dimension mapping dimensions_mapping = {
    "Organisation": "hierarchy_1", "Location": ["hierarchy_2", "hierarchy_3", "hierarchy_4"]
}
dataframe = pd.DataFrame(data)
with TM1Service(**tm1params) as tm1:
    tm1.cells.clear_with_dataframe(cube="Sales", df=dataframe)
```

clear_with_mdx(*cube: str*, *mdx: str*, *sandbox_name: str = None*, ***kwargs*)

clear a slice in a cube based on an MDX query. Function requires admin permissions, since TM1py uses an unbound TI with a *ViewZeroOut* statement.

Parameters

- **cube** – name of the cube
- **mdx** – a valid MDX query
- **sandbox_name** – a valid existing sandbox for the current user
- **kwargs** –

Returns

create_cellset(*mdx: str | MdxBuilder*, *sandbox_name: str = None*, ***kwargs*) → str

Execute MDX in order to create cellset at server. return the cellset-id

Parameters

- **mdx** – MDX Query, as string
- **sandbox_name** – str

Returns

create_cellset_from_view(*cube_name: str*, *view_name: str*, *private: bool*, *sandbox_name: str = None*, ***kwargs*) → str

create cellset from a cube view. return the cellset-id

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **kwargs** –
- **sandbox_name** – str

Returns

deactivate_transactionlog(**args: str*, ***kwargs*) → Response

Deactivate Transactionlog for one or many cubes

Parameters
args – one or many cube names

Returns

delete_cellset(*cellset_id: str*, *sandbox_name: str = None*, ***kwargs*) → Response

Delete a cellset

Parameters

- **cellset_id** –
- **sandbox_name** – str

Returns

drop_non_updateable_cells(*cells: Dict*, *cube_name: str*, *dimensions: List[str]*)

end_changeset(*change_set: str*) → Response

end a change set

Returns
Change set ID

execute_mdx(*mdx: str*, *cell_properties: List[str] = None*, *top: int = None*, *skip_contexts: bool = False*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *element_unique_names: bool = True*, *skip_cell_properties: bool = False*, *use_compact_json: bool = False*, *skip_sandbox_dimension: bool = False*, *max_workers: int = 1*, *async_axis: int = 0*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

Execute MDX and return the cells with their properties

Parameters

- **mdx** – MDX Query, as string
- **cell_properties** – properties to be queried from the cell. E.g. Value, Ordinal, RuleDerived, …
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_contexts** – skip elements from titles / contexts in response
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **element_unique_names** – '[d1].[h1].[e1]' or 'e1'
- **skip_cell_properties** – cell values in result dictionary, instead of cell_properties dictionary
- **use_compact_json** – bool

Skip_sandbox_dimension
bool = False

Returns
content in sweet concise structure.

execute_mdx_async(*mdx: str*, *cell_properties: List[str] = None*, *top: int = None*, *skip_contexts: bool = False*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *element_unique_names: bool = True*, *skip_cell_properties: bool = False*, *use_compact_json: bool = False*, *skip_sandbox_dimension: bool = False*, *max_workers: int = 8*, *async_axis: int = 0*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

Execute MDX and return the cells with their properties

Parameters

- **mdx** – MDX Query, as string
- **cell_properties** – properties to be queried from the cell. E.g. Value, Ordinal, RuleDerived, …
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_contexts** – skip elements from titles / contexts in response
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)

- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **element_unique_names** – '[d1].[h1].[e1]' or 'e1'
- **skip_cell_properties** – cell values in result dictionary, instead of cell_properties dictionary
- **use_compact_json** – bool
- **skip_sandbox_dimension** – bool = False
- **max_workers** – Int, number of threads to use in parallel
- **async_axis** – 0 (columns) or 1 (rows). On which axis to parallelize retrieval

Returns
content in sweet concise structure.

execute_mdx_cellcount(*mdx: str*, *sandbox_name: str = None*, ***kwargs*) → int

Execute MDX in order to understand how many cells are in a cellset. Only return number of cells in the cellset. FAST!

Parameters

- **mdx** – MDX Query, as string
- **sandbox_name** – str

Returns
Number of Cells in the CellSet

execute_mdx_csv(*mdx: str | MdxBuilder*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *csv_dialect: Dialect = None*, *line_separator: str = '\r\n'*, *value_separator: str = ','*, *sandbox_name: str = None*, *include_attributes: bool = False*, *use_iterative_json: bool = False*, *use_compact_json: bool = False*, *use_blob: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → str

Optimized for performance. Get csv string of coordinates and values.

Parameters

- **mdx** – Valid MDX Query
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **csv_dialect** – provide all csv output settings through standard library csv.Dialect If not provided dialect is created based on line_separator and value_separator arguments.
- **line_separator** –
- **value_separator** –
- **sandbox_name** – str
- **include_attributes** – include attribute columns

- **use_iterative_json** – use iterative json parsing to reduce memory consumption significantly.

Comes at a cost of 3-5% performance. :param use_compact_json: bool :param use_blob: Has better performance on datasets > 1M cells and lower memory footprint in any case. :param mdx_headers: boolean, fully qualified hierarchy name as header instead of simple dimension name :return: String

execute_mdx_dataframe(*mdx: str | MdxBuilder*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *include_attributes: bool = False*, *use_iterative_json: bool = False*, *use_compact_json: bool = False*, *use_blob: bool = False*, *shaped: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → DataFrame

Optimized for performance. Get Pandas DataFrame from MDX Query.

Takes all arguments from the pandas.read_csv method: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

If 'use_blob' and 'shaped' are True, 'skip_zeros' will be overruled to False. This is necessary to assure column order is in line with cube view in TM1

Parameters

- **mdx** – Valid MDX Query
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **include_attributes** – include attribute columns
- **use_iterative_json** – use iterative json parsing to reduce memory consumption significantly.

Comes at a cost of 3-5% performance. :param use_compact_json: bool :param use_blob: Has better performance on datasets > 1M cells and lower memory footprint in any case. :param shaped: preserve shape of view/mdx in data frame :param mdx_headers: boolean, fully qualified hierarchy name as header instead of simple dimension name :return: Pandas Dataframe

execute_mdx_dataframe_async(*mdx_list: List[str | MdxBuilder]*, *max_workers: int = 8*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *include_attributes: bool = False*, *use_iterative_json: bool = False*, *use_compact_json: bool = False*, *use_blob: bool = False*, *shaped: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → DataFrame

execute_mdx_dataframe_pivot(*mdx: str*, *dropna: bool = False*, *fill_value: bool = None*, *sandbox_name: str = None*) → DataFrame

Execute MDX Query to get a pandas pivot data frame in the shape as specified in the Query

Parameters

- **mdx** –
- **dropna** –

- **fill_value** –
- **sandbox_name** – str

Returns

execute_mdx_dataframe_shaped(*mdx: str*, *sandbox_name: str = None*, *display_attribute: bool = False*, *use_iterative_json: bool = False*, *use_blob: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → DataFrame

Retrieves data from cube in the shape of the query. Dimensions on rows can be stacked. One dimension must be placed on columns. Title selections are ignored.

Parameters

- **mdx** –
- **sandbox_name** – str

:param use_blob :param use_iterative_json :param display_attribute: bool, show element name or first attribute from MDX PROPERTIES clause :param kwargs: :return:

execute_mdx_elements_value_dict(*mdx: str*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *element_separator: str = '|'*, *sandbox_name: str = None*, ***kwargs*) → CaseAndSpaceInsensitiveDict

Optimized for performance. Get Dict from MDX Query. :param mdx: Valid MDX Query :param top: Int, number of cells to return (counting from top) :param skip: Int, number of cells to skip (counting from top) :param skip_zeros: skip zeros in cellset (irrespective of zero suppression in MDX / view) :param skip_consolidated_cells: skip consolidated cells in cellset :param skip_rule_derived_cells: skip rule derived cells in cellset :param element_separator: separator for the dimension element combination :param sandbox_name: str :return: CaseAndSpaceInsensitiveDict {'2020|Jan|Sales': 2000, '2020|Feb|Sales': 3000}

execute_mdx_raw(*mdx: str*, *cell_properties: Iterable[str] = None*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *top: int = None*, *skip_contexts: bool = False*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *include_hierarchies: bool = False*, *use_compact_json: bool = False*, ***kwargs*) → Dict

Execute MDX and return the raw data from TM1

Parameters

- **mdx** – String, a valid MDX Query
- **cell_properties** – List of properties to be queried from the cell. E.g. ['Value', 'RuleDerived', ...]
- **elem_properties** – List of properties to be queried from the elements. E.g. ['Name','Attributes', ...]
- **member_properties** – List of properties to be queried from the members. E.g. ['Name','Attributes', ...]
- **top** – Integer limiting the number of cells and the number or rows returned
- **skip** – Integer limiting the number of cells and the number or rows returned
- **skip_contexts** – skip elements from titles / contexts in response
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset

- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **include_hierarchies** – retrieve Hierarchies property on Axes
- **use_compact_json** – bool

Returns
Raw format from TM1.

execute_mdx_rows_and_values(*mdx: str*, *element_unique_names: bool = True*, *sandbox_name: str = None*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

Execute MDX and retrieve row element names and values in a case and space insensitive dictionary

Parameters

- **mdx** –
- **element_unique_names** –
- **sandbox_name** – str
- **kwargs** –

Returns

execute_mdx_rows_and_values_string_set(*mdx: str*, *exclude_empty_cells: bool = True*, *sandbox_name: str = None*, ***kwargs*) → CaseAndSpaceInsensitiveSet

Retrieve row element names and **string** cell values in a case and space insensitive set

Parameters

- **exclude_empty_cells** –
- **mdx** –
- **sandbox_name** – str

Returns

execute_mdx_ui_array(*mdx: str*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *value_precision: int = 2*, *top: int = None*, *skip: int = None*, *sandbox_name: str = None*, *use_compact_json: bool = False*, ***kwargs*)

Useful for grids or charting libraries that want an array of cell values per row. Returns 3-dimensional cell structure for tabbed grids or multiple charts. Rows and pages are dicts, addressable by their name. Proper order of rows can be obtained in headers[1] Example 'cells' return format:

'cells': {

'10100': {

'Net Operating Income': [19832724.72429739,
20365654.788303416, 20729201.329183243, 20480205.20121749],

'Revenue': [28981046.50724231,
29512482.207418434, 29913730.038971487, 29563345.9542385]},

'10200': {

'Net Operating Income': [9853293.623709997,
10277650.763958748, 10466934.096533755, 10333095.839474997],

'Revenue': [13888143.710000003,
14300216.43, 14502421.63, 14321501.940000001]}

},

Parameters

- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **mdx** – a valid MDX Query
- **elem_properties** – List of properties to be queried from the elements. E.g. ['UniqueName','Attributes']
- **member_properties** – List of properties to be queried from the members. E.g. ['UniqueName','Attributes']
- **value_precision** – Integer (optional) specifying number of decimal places to return
- **sandbox_name** – str
- **use_compact_json** – bool

Returns

dict :{ titles: [], headers: [axis][], cells:{ Page0:{ Row0:{ [row values], Row1: [], ... }, ... }, ... }}

execute_mdx_ui_dygraph(*mdx: str*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *value_precision: int = 2*, *top: int = None*, *skip: int = None*, *sandbox_name: str = None*, *use_compact_json: bool = False*, ***kwargs*) → Dict

Execute MDX get dygraph dictionary Useful for grids or charting libraries that want an array of cell values per column Returns 3-dimensional cell structure for tabbed grids or multiple charts Example 'cells' return format:

'cells': {

'10100': [
['Q1-2004', 28981046.50724231, 19832724.72429739], ['Q2-2004', 29512482.207418434, 20365654.788303416], ['Q3-2004', 29913730.038971487, 20729201.329183243], ['Q4-2004', 29563345.9542385, 20480205.20121749]],

'10200': [
['Q1-2004', 13888143.710000003, 9853293.623709997], ['Q2-2004', 14300216.43, 10277650.763958748], ['Q3-2004', 14502421.63, 10466934.096533755], ['Q4-2004', 14321501.940000001, 10333095.839474997]]

},

Parameters

- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **mdx** – String, valid MDX Query
- **elem_properties** – List of properties to be queried from the elements. E.g. ['UniqueName','Attributes']
- **member_properties** – List of properties to be queried from the members. E.g. ['UniqueName','Attributes']
- **value_precision** – Integer (optional) specifying number of decimal places to return

- **sandbox_name** – str
- **use_compact_json** – bool

Returns
dict: { titles: [], headers: [axis][], cells: { Page0: [[column name, column values], [], …], … }}

execute_mdx_values(*mdx: str*, *sandbox_name: str = None*, *use_compact_json: bool = False*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, ***kwargs*) → List[str | float]

Optimized for performance. Query only raw cell values. Coordinates are omitted !

Parameters

- **mdx** – a valid MDX Query
- **sandbox_name** – str
- **use_compact_json** – bool
- **skip_zeros** – bool
- **skip_consolidated_cells** – bool
- **skip_rule_derived_cells** – bool

Returns
List of cell values

execute_unbound_process(*process:* Process, ***kwargs*) → Tuple[bool, str, str]

execute_view(*cube_name: str*, *view_name: str*, *private: bool = False*, *cell_properties: Iterable[str] = None*, *top: int = None*, *skip_contexts: bool = False*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *element_unique_names: bool = True*, *skip_cell_properties: bool = False*, *use_compact_json: bool = False*, *max_workers: int = 1*, *async_axis: int = 0*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

get view content as dictionary with sweet and concise structure.
Works on NativeView and MDXView !

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **cell_properties** – List, cell properties: [Values, Status, HasPicklist, etc.]
- **private** – Boolean
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_contexts** – skip elements from titles / contexts in response
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset

- **element_unique_names** – '[d1].[h1].[e1]' or 'e1'
- **sandbox_name** – str
- **skip_cell_properties** – cell values in result dictionary, instead of cell_properties dictionary
- **max_workers** – Int, number of threads to use in parallel
- **async_axis** – 0 (columns) or 1 (rows). On which axis to parallelize retrieval
- **use_compact_json** – bool

Returns
Dictionary : {([dim1].[elem1], [dim2][elem6]): {'Value':3127.312, 'Ordinal':12} .... }

execute_view_async(*cube_name: str*, *view_name: str*, *private: bool = False*, *cell_properties: Iterable[str] = None*, *top: int = None*, *skip_contexts: bool = False*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *element_unique_names: bool = True*, *skip_cell_properties: bool = False*, *max_workers: int = 8*, *async_axis: int = 0*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

get view content as dictionary with sweet and concise structure.
Works on NativeView and MDXView !

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **cell_properties** – List, cell properties: [Values, Status, HasPicklist, etc.]
- **private** – Boolean
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_contexts** – skip elements from titles / contexts in response
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **element_unique_names** – '[d1].[h1].[e1]' or 'e1'
- **sandbox_name** – str
- **skip_cell_properties** – cell values in result dictionary, instead of cell_properties dictionary
- **max_workers** – Int, number of threads to use in parallel
- **async_axis** – 0 (columns) or 1 (rows). On which axis to parallelize retrieval

Returns
Dictionary : {([dim1].[elem1], [dim2][elem6]): {'Value':3127.312, 'Ordinal':12} .... }

execute_view_cellcount(*cube_name: str*, *view_name: str*, *private: bool = False*, *sandbox_name: str = None*, ***kwargs*) → int

Execute cube view in order to understand how many cells are in a cellset. Only return number of cells in the cellset. FAST!

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **sandbox_name** – str

Returns

execute_view_csv(*cube_name: str*, *view_name: str*, *private: bool = False*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *csv_dialect: Dialect = None*, *line_separator: str = \r\n'*, *value_separator: str = ','*, *sandbox_name: str = None*, *use_iterative_json: bool = False*, *use_compact_json: bool = False*, *use_blob: bool = False*, *arranged_axes: Tuple[List, List, List] = None*, *mdx_headers: bool = False*, ***kwargs*) → str

Optimized for performance. Get csv string of coordinates and values.

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **csv_dialect** – provide all csv output settings through standard library csv.Dialect If not provided dialect is created based on line_separator and value_separator arguments.
- **line_separator** –
- **value_separator** –
- **sandbox_name** – str
- **use_iterative_json** – use iterative json parsing to reduce memory consumption significantly.

Comes at a cost of 3-5% performance. :param use_compact_json: bool :param use_blob: Has 40% better performance and lower memory footprint in any case. Requires admin permissions. :param arranged_axes: Tuple of dimension names on all axes as 3 lists: Titles, Rows, Columns.

Allows function to skip retrieval of cellset composition. E.g.: arranged_axes=([“Year”], [“Region”,”Product”], [“Period”, “Version”])

Parameters

mdx_headers – boolean, fully qualified hierarchy name as header instead of simple dimension name

Returns
String

execute_view_dataframe(*cube_name: str*, *view_name: str*, *private: bool = False*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *use_iterative_json: bool = False*, *use_blob: bool = False*, *shaped: bool = False*, *arranged_axes: Tuple[List, List, List] = None*, *mdx_headers: bool = False*, ***kwargs*) → DataFrame

Optimized for performance. Get Pandas DataFrame from an existing Cube View Context dimensions are omitted in the resulting Dataframe ! Cells with Zero/null are omitted !

If 'use_blob' and 'shaped' are True, 'skip_zeros' will be overruled to False. This is necessary to assure column order is in line with cube view in TM1

Takes all arguments from the pandas.read_csv method: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **use_iterative_json** – use iterative json parsing to reduce memory consumption significantly.

Comes at a cost of 3-5% performance. :param use_blob: Has 40% better performance and lower memory footprint in any case. Requires admin permissions. :param shaped: Shape rows and columns of data frame as specified in cube view / MDX :param arranged_axes: Tuple of dimension names on all axes as 3 lists: Titles, Rows, Columns.

Allows function to skip retrieval of cellset composition in use_blob mode. E.g.: axes=(["Year"], ["Region","Product"], ["Period", "Version"]) :param mdx_headers: boolean, fully qualified hierarchy name as header instead of simple dimension name

Returns
Pandas Dataframe

execute_view_dataframe_pivot(*cube_name: str*, *view_name: str*, *private: bool = False*, *dropna: bool = False*, *fill_value: bool = None*, *sandbox_name: str = None*, ***kwargs*) → DataFrame

Execute a cube view to get a pandas pivot dataframe, in the shape of the cube view

Parameters

- **cube_name** – String, name of the cube

- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **dropna** –
- **fill_value** –
- **sandbox_name** – str

Returns

execute_view_dataframe_shaped(*cube_name: str*, *view_name: str*, *private: bool = False*, *sandbox_name: str = None*, *use_iterative_json: bool = False*, *use_blob: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → DataFrame

Retrieves data from cube in the shape of the query. Dimensions on rows can be stacked. One dimension must be placed on columns. Title selections are ignored.

Parameters

- **cube_name** –
- **view_name** –
- **private** –
- **sandbox_name** – str

:param use_blob :param use_iterative_json :param kwargs: :return:

execute_view_elements_value_dict(*cube_name: str*, *view_name: str*, *private: bool = False*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *element_separator: str = '|'*, *sandbox_name: str = None*, ***kwargs*) → CaseAndSpaceInsensitiveDict

Optimized for performance. Get a Dict(tuple, value) from an existing Cube View Context dimensions are omitted in the resulting Dataframe ! Cells with Zero/null are omitted by default, but still configurable!

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **element_separator** – separator for the dimension element combination
- **sandbox_name** – str

Returns

CaseAndSpaceInsensitiveDict {'2020|Jan|Sales': 2000, '2020|Feb|Sales': 3000}

execute_view_raw(*cube_name: str*, *view_name: str*, *private: bool = False*, *cell_properties: Iterable[str] = None*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *top: int = None*, *skip_contexts: bool = False*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *use_compact_json: bool = False*, ***kwargs*) → Dict

Execute a cube view and return the raw data from TM1

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **cell_properties** – List of properties to be queried from the cell. E.g. ['Value', 'RuleDerived', ...]
- **elem_properties** – List of properties to be queried from the elements. E.g. ['Name','Attributes', ...]
- **member_properties** – List of properties to be queried from the members. E.g. ['Name','Attributes', ...]
- **top** – Integer limiting the number of cells and the number or rows returned
- **skip_contexts** – skip elements from titles / contexts in response
- **skip** – Integer limiting the number of cells and the number or rows returned
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **use_compact_json** – bool

Returns

Raw format from TM1.

execute_view_rows_and_values(*cube_name: str*, *view_name: str*, *private: bool = False*, *element_unique_names: bool = True*, *sandbox_name: str = None*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

Execute cube view and retrieve row element names and values in a case and space insensitive dictionary

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **element_unique_names** –
- **sandbox_name** – str
- **kwargs** –

Returns

execute_view_rows_and_values_string_set(*cube_name: str*, *view_name: str*, *private: bool = False*, *exclude_empty_cells: bool = True*, *sandbox_name: str = None*, ***kwargs*) → CaseAndSpaceInsensitiveSet

Retrieve row element names and **string** cell values in a case and space insensitive set

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **exclude_empty_cells** –
- **sandbox_name** – str

Returns

execute_view_ui_array(*cube_name: str*, *view_name: str*, *private: bool = False*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *value_precision: int = 2*, *top: int = None*, *skip: int = None*, *sandbox_name: str = None*, *use_compact_json: bool = False*, ***kwargs*)

Useful for grids or charting libraries that want an array of cell values per row. Returns 3-dimensional cell structure for tabbed grids or multiple charts. Rows and pages are dicts, addressable by their name. Proper order of rows can be obtained in headers[1] Example 'cells' return format:

'cells': {

'10100': {

'Net Operating Income': [19832724.72429739,
20365654.788303416, 20729201.329183243, 20480205.20121749],

'Revenue': [28981046.50724231,
29512482.207418434, 29913730.038971487, 29563345.9542385]},

'10200': {

'Net Operating Income': [9853293.623709997,
10277650.763958748, 10466934.096533755, 10333095.839474997],

'Revenue': [13888143.710000003,
14300216.43, 14502421.63, 14321501.940000001]}

},

Parameters

- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **elem_properties** – List of properties to be queried from the elements. E.g. ['UniqueName','Attributes']
- **member_properties** – List properties to be queried from the member. E.g. ['Name', 'UniqueName']

- **value_precision** – Integer (optional) specifying number of decimal places to return
- **sandbox_name** – str
- **use_compact_json** – bool

Returns

dict :{ titles: [], headers: [axis][], cells:{ Page0:{ Row0: {[row values], Row1: [], ... }, ... }, ... }}

execute_view_ui_dygraph(*cube_name: str*, *view_name: str*, *private: bool = False*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *value_precision: int = 2*, *top: int = None*, *skip: int = None*, *sandbox_name: str = None*, *use_compact_json: bool = False*, ***kwargs*)

Useful for grids or charting libraries that want an array of cell values per row. Returns 3-dimensional cell structure for tabbed grids or multiple charts. Rows and pages are dicts, addressable by their name. Proper order of rows can be obtained in headers[1] Example 'cells' return format:

'cells': {

'10100': {

'Net Operating Income': [19832724.72429739,
20365654.788303416, 20729201.329183243, 20480205.20121749],

'Revenue': [28981046.50724231,
29512482.207418434, 29913730.038971487, 29563345.9542385]},

'10200': {

'Net Operating Income': [9853293.623709997,
10277650.763958748, 10466934.096533755, 10333095.839474997],

'Revenue': [13888143.710000003,
14300216.43, 14502421.63, 14321501.940000001]}

},

Parameters

- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **cube_name** – cube name
- **view_name** – view name
- **private** – True (private) or False (public)
- **elem_properties** – List of properties to be queried from the elements. E.g. ['UniqueName','Attributes']
- **member_properties** – List of properties to be queried from the members. E.g. ['UniqueName','Attributes']
- **value_precision** – Integer (optional) specifying number of decimal places to return
- **sandbox_name** – str
- **use_compact_json** – bool

Returns

execute_view_values(*cube_name: str*, *view_name: str*, *private: bool = False*, *sandbox_name: str = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *use_compact_json: bool = False*, ***kwargs*) → List[str | float]

Execute view and retrieve only the cell values

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – True (private) or False (public)
- **sandbox_name** – str
- **use_compact_json** – bool
- **skip_zeros** – bool
- **skip_consolidated_cells** – bool
- **skip_rule_derived_cells** – bool
- **kwargs** –

Returns

extract_cellset(*cellset_id: str*, *cell_properties: Iterable[str] = None*, *top: int = None*, *skip: int = None*, *delete_cellset: bool = True*, *skip_contexts: bool = False*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *element_unique_names: bool = True*, *skip_cell_properties: bool = False*, *use_compact_json: bool = False*, *skip_sandbox_dimension: bool = False*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

Execute cellset and return the cells with their properties

Parameters

- **skip_contexts** –
- **delete_cellset** –
- **cellset_id** –
- **cell_properties** – properties to be queried from the cell. E.g. Value, Ordinal, RuleDerived, …
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **element_unique_names** – '[d1].[h1].[e1]' or 'e1'
- **skip_cell_properties** – cell values in result dictionary, instead of cell_properties dictionary
- **use_compact_json** – bool
- **skip_sandbox_dimension** – skip sandbox dimension

Returns
Content in sweet concise structure.

extract_cellset_async(*cellset_id: str*, *cell_properties: Iterable[str] = None*, *top: int = None*, *skip: int = None*, *delete_cellset: bool = True*, *skip_contexts: bool = False*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *element_unique_names: bool = True*, *skip_cell_properties: bool = False*, *skip_sandbox_dimension: bool = False*, *max_workers: int = 8*, *async_axis: int = 1*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

Execute cellset and return the cells with their properties

Parameters

- **skip_contexts** –
- **delete_cellset** –
- **cellset_id** –
- **cell_properties** – properties to be queried from the cell. E.g. Value, Ordinal, RuleDerived, ...
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **element_unique_names** – '[d1].[h1].[e1]' or 'e1'
- **skip_cell_properties** – cell values in result dictionary, instead of cell_properties dictionary
- **skip_sandbox_dimension** – skip sandbox dimension
- **max_workers** – Int, number of threads to use in parallel
- **async_axis** – 0 (columns) or 1 (rows). On which axis to parallelize retrieval

Returns
Content in sweet concise structure.

extract_cellset_axes_cardinality(*cellset_id: str*)

extract_cellset_axes_raw_async(*cellset_id: str*, *async_axis: int = 1*, *max_workers: int = 8*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *skip_contexts: bool = False*, *include_hierarchies: bool = False*, *sandbox_name: str = None*, ***kwargs*)

Extract cellset axes asynchronously

Parameters

- **cellset_id** – String; ID of existing cellset
- **async_axis** – determines which axis will be extracted asynchronously
- **max_workers** – Max number of threads, e.g. 14

- **elem_properties** – List of properties to be queried from elements. E.g. ['UniqueName','Attributes', ...]
- **member_properties** – List properties to be queried from the member. E.g. ['Name', 'UniqueName']
- **skip_contexts** – skip elements from titles / contexts in response
- **sandbox_name** – str
- **include_hierarchies** – retrieve Hierarchies property on Axes

Returns
Raw format from TM1.

extract_cellset_cellcount(*cellset_id: str*, *sandbox_name: str = None*, ***kwargs*) → int

Retrieve number of cells in the cellset

Parameters

- **cellset_id** –
- **sandbox_name** – str
- **kwargs** –

Returns

extract_cellset_cells_raw(*cellset_id: str*, *cell_properties: Iterable[str] = None*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, ***kwargs*)

extract_cellset_cells_raw_async(*cellset_id: str*, *max_workers: int = 8*, *cell_properties: Iterable[str] = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, ***kwargs*)

extract_cellset_composition(*cellset_id: str*, *sandbox_name: str = None*, ***kwargs*) → Tuple[str, List[str], List[str], List[str]]

Retrieve composition of dimensions on the axes in the cellset

Parameters

- **cellset_id** –
- **kwargs** –
- **sandbox_name** – str

Returns

extract_cellset_csv(*cellset_id: str*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *csv_dialect: Dialect = None*, *line_separator: str = '\r\n'*, *value_separator: str = ','*, *sandbox_name: str = None*, *include_attributes: bool = False*, *use_compact_json: bool = False*, *include_headers: bool = True*, *mdx_headers: bool = False*, ***kwargs*) → str

Execute cellset and return only the 'Content', in csv format

Parameters

- **cellset_id** – String; ID of existing cellset

- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **csv_dialect** – provide all csv output settings through standard library csv.Dialect If not provided dialect is created based on line_separator and value_separator arguments.
- **line_separator** –

:param value_separator :param sandbox_name: str :param include_attributes: include attribute columns :param use_compact_json: boolean :param include_headers: boolean :param mdx_headers: boolean. Fully qualified hierarchy name as header instead of simple dimension name :return: Raw format from TM1.

extract_cellset_csv_iter_json(*cellset_id: str*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *csv_dialect: Dialect = None*, *line_separator: str = '\r\n'*, *value_separator: str = ','*, *sandbox_name: str = None*, *include_attributes: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → str

Execute cellset and return only the 'Content', in csv format

Parameters

- **cellset_id** – String; ID of existing cellset
- **top** – Int, number of cells to return (counting from top)
- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **csv_dialect** – provide all csv output settings through standard library csv.Dialect If not provided dialect is created based on line_separator and value_separator arguments.
- **line_separator** –

:param value_separator :param sandbox_name: str :param include_attributes: boolean :param mdx_headers: boolean. Fully qualified hierarchy name as header instead of simple dimension name :return: Raw format from TM1.

extract_cellset_cube_with_dimensions(*cellset_id: str*, ***kwargs*)

extract_cellset_dataframe(*cellset_id: str*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = True*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *include_attributes: bool = False*, *use_iterative_json: bool = False*, *use_compact_json: bool = False*, *shaped: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → DataFrame

Build pandas data frame from cellset_id

Parameters

- **cellset_id** –
- **top** – Int, number of cells to return (counting from top)

- **skip** – Int, number of cells to skip (counting from top)
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **include_attributes** – include attribute columns
- **use_iterative_json** – use iterative json parsing to reduce memory consumption significantly.

Comes at a cost of 3-5% performance. :param use_compact_json: bool :param kwargs: :return:

extract_cellset_dataframe_pivot(*cellset_id: str*, *dropna: bool = False*, *fill_value: bool = False*, *sandbox_name: str = None*, *use_compact_json: bool = False*, ***kwargs*) → DataFrame

Extract a pivot table (pandas dataframe) from a cellset in TM1

Parameters

- **cellset_id** –
- **dropna** –
- **fill_value** –
- **kwargs** –
- **sandbox_name** – str
- **use_compact_json** – bool

Returns

extract_cellset_dataframe_shaped(*cellset_id: str*, *sandbox_name: str = None*, *display_attribute: bool = False*, *infer_dtype: bool = False*, *mdx_headers: bool = False*, ***kwargs*) → DataFrame

Retrieves data from cellset in the shape of the query. Dimensions on rows can be stacked. One dimension must be placed on columns. Title selections are ignored.

:param cellset_id :param sandbox_name: str :param display_attribute: bool, show element name or first attribute from MDX PROPERTIES clause :param infer_dtype: bool, if True, lets pandas infer dtypes, otherwise all columns will be of type str.

extract_cellset_metadata_raw(*cellset_id: str*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *top: int = None*, *skip: int = None*, *skip_contexts: bool = False*, *include_hierarchies: bool = False*, *sandbox_name: str = None*, ***kwargs*)

extract_cellset_partition(*cellset_id: str*, *partition_start_ordinal: int*, *partition_end_ordinal: int*, *cell_properties: Iterable[str] = None*, *top: int = None*, *skip: int = None*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*) → Dict

Method to extract a cellset partition. Cellset partitions are a collection of cellset cells where they have a defined top left boundary, and bottom right boundary. Read More: https://www.ibm.com/docs/en/planning-analytics/2.0.0?topic=data-cellsets#dg_tm1_odata_get_cells__title__1 :param partition_start_ordinal: top left cell boundary :param partition_end_ordinal: bottom right cell boundary :param cell_properties: cell properties to include, default: Orginal, Value :param top: Integer limiting the number of cells and the number or rows returned :param skip: Integer limiting the number of cells and the number

or rows returned :param skip_zeros: skip zeros in cellset (irrespective of zero suppression in MDX / view) :param skip_consolidated_cells: skip consolidated cells in cellset :param skip_rule_derived_cells: skip rule derived cells in cellset :param sandbox_name: str :return: CellSet Dictionary

extract_cellset_raw(*cellset_id: str*, *cell_properties: Iterable[str] = None*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *top: int = None*, *skip: int = None*, *skip_contexts: bool = False*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *include_hierarchies: bool = False*, *use_compact_json: bool = False*, ***kwargs*) → Dict

Extract full cellset data and return the raw data from TM1

Parameters

- **cellset_id** – String; ID of existing cellset
- **cell_properties** – List of properties to be queried from cells. E.g. [‘Value’, ‘RuleDerived’, . . .]
- **elem_properties** – List of properties to be queried from elements. E.g. [‘UniqueName’,’Attributes’, . . .]
- **member_properties** – List properties to be queried from the member. E.g. [‘Name’, ‘UniqueName’]
- **top** – Integer limiting the number of cells and the number or rows returned
- **skip** – Integer limiting the number of cells and the number or rows returned
- **skip_contexts** –
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **include_hierarchies** – retrieve Hierarchies property on Axes
- **use_compact_json** – bool

Returns

Raw format from TM1.

extract_cellset_raw_response(*cellset_id: str*, *cell_properties: Iterable[str] = None*, *elem_properties: Iterable[str] = None*, *member_properties: Iterable[str] = None*, *top: int = None*, *skip: int = None*, *skip_contexts: bool = False*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, *sandbox_name: str = None*, *include_hierarchies: bool = False*, ***kwargs*) → Response

Extract full cellset data and return the raw data from TM1

Parameters

- **cellset_id** – String; ID of existing cellset
- **cell_properties** – List of properties to be queried from cells. E.g. [‘Value’, ‘RuleDerived’, . . .]
- **elem_properties** – List of properties to be queried from elements. E.g. [‘UniqueName’,’Attributes’, . . .]

- **member_properties** – List properties to be queried from the member. E.g. ['Name', 'UniqueName']
- **top** – Integer limiting the number of cells and the number or rows returned
- **skip** – Integer limiting the number of cells and the number or rows returned
- **skip_contexts** –
- **skip_zeros** – skip zeros in cellset (irrespective of zero suppression in MDX / view)
- **skip_consolidated_cells** – skip consolidated cells in cellset
- **skip_rule_derived_cells** – skip rule derived cells in cellset
- **sandbox_name** – str
- **include_hierarchies** – retrieve Hierarchies property on Axes

Returns
Raw format from TM1.

extract_cellset_rows_and_values(*cellset_id: str*, *element_unique_names: bool = True*, *sandbox_name: str = None*, ***kwargs*) → CaseAndSpaceInsensitiveTuplesDict

Retrieve row element names and values in a case and space insensitive dictionary

Parameters

- **cellset_id** –
- **element_unique_names** –
- **kwargs** –
- **sandbox_name** – str

Returns

extract_cellset_values(*cellset_id: str*, *sandbox_name: str = None*, *use_compact_json: bool = False*, *skip_zeros: bool = False*, *skip_consolidated_cells: bool = False*, *skip_rule_derived_cells: bool = False*, ***kwargs*) → List[str | float]

Extract cellset data and return only the cells and values

Parameters

- **cellset_id** – String; ID of existing cellset
- **sandbox_name** – str
- **use_compact_json** – bool
- **skip_zeros** – bool
- **skip_consolidated_cells** – bool
- **skip_rule_derived_cells** – bool

Returns
Raw format from TM1.

generate_enable_sandbox_ti(*sandbox_name*)

get_cellset_cells_count(*mdx: str*) → int

Execute MDX in order to understand how many cells are in a cellset

Parameters
mdx – MDX Query, as string

Returns
Number of Cells in the CellSet

`get_cube_service`()

`get_dimension_names_for_writing`(*cube_name: str*, ***kwargs*) → List[str]

Get dimensions of a cube. Skip sandbox dimension

Parameters

- **cube_name** –
- **kwargs** –

Returns

`get_element_service`()

`get_elements_from_all_measure_hierarchies`(*cube_name: str*) → Dict[str, str]

`get_error_log_file_content`(*file_name: str*, ***kwargs*) → str

`get_value`(*cube_name: str*, *elements: str | Iterable = None*, *dimensions: List[str] = None*, *sandbox_name: str = None*, *element_separator: str = ','*, *hierarchy_separator: str = '&&'*, *hierarchy_element_separator: str = '::'*, ***kwargs*) → str | float

Returns cube value from specified coordinates

Parameters

- **cube_name** – Name of the cube
- **elements** – Describes the Dimension-Hierarchy-Element arrangement - Example: "Hierarchy1::Element1 && Hierarchy2::Element4, Element9, Element2" - Dimensions are not specified! They are derived from the position. - The , separates the element-selections - If more than one hierarchy is selected per dimension && splits the elementselections - If no Hierarchy is specified. Default Hierarchy will be addressed

or Iterable of type mdxpy.Member or similar

- Dimension names must be provided in this case! Example: [(Dimension1, Element1), (Dimension2, Element2), (Dimension3, Element3)]
- Hierarchys can be included. Example: [(Dimension1, Hierarchy1, Element1), (Dimension1, Hierarchy2, Element2), (Dimension2, Element3)]

Parameters

- **dimensions** – List of dimension names in correct order
- **sandbox_name** – str
- **element_separator** – Alternative separator for the element selections
- **hierarchy_separator** – Alternative separator for multiple hierarchies
- **hierarchy_element_separator** – Alternative separator between hierarchy name and element name

Returns

get_values(*cube_name: str*, *element_sets: Iterable[Iterable[str]] = None*, *dimensions: List[str] = None*, *sandbox_name: str = None*, *element_separator: str = ','*, *hierarchy_separator: str = '&&'*, *hierarchy_element_separator: str = '::'*, ***kwargs*) → List

Returns list of cube values from specified coordinates list. will be in same order as original list

Parameters

- **cube_name** – Name of the cube
- **element_sets** – Set of coordinates where each element is provided in the correct dimension order.

[('2024', 'Actual', 'London', 'P02), ('2024', 'Forecast', 'Berlin', 'P03)] :param dimensions: Dimension names in correct order :param sandbox_name: str :param element_separator: Alternative separator for the element selections :param hierarchy_separator: Alternative separator for multiple hierarchies :param hierarchy_element_separator: Alternative separator between hierarchy name and element name :return:

get_view_content(*cube_name: str*, *view_name: str*, *cell_properties: Iterable[str] = None*, *private: bool = False*, *top: int = None*)

relative_proportional_spread(*value: float*, *cube: str*, *unique_element_names: Iterable[str]*, *reference_unique_element_names: Iterable[str]*, *reference_cube: str = None*, *sandbox_name: str = None*, ***kwargs*) → Response

Execute relative proportional spread

Parameters

- **value** – value to be spread
- **cube** – name of the cube
- **unique_element_names** – target cell coordinates as unique element names (e.g. ["[d1].[c1]","[d2].[e3]"])
- **reference_cube** – name of the reference cube. Can be None
- **reference_unique_element_names** – reference cell coordinates as unique element names
- **sandbox_name** – str

Returns

sandbox_exists(*sandbox_name*) → bool

trace_cell_calculation(*cube_name: str*, *elements: Iterable | str*, *dimensions: Iterable[str] = None*, *sandbox_name: str = None*, *depth: int = 1*, *element_separator: str = ','*, *hierarchy_separator: str = '&&'*, *hierarchy_element_separator: str = '::'*, ***kwargs*) → Dict

Trace cell calculation at specified coordinates

Parameters

- **cube_name** – name of the target cube
- **elements** –

string "Hierarchy1::Element1 && Hierarchy2::Element4, Element9, Element2"

- Dimensions are not specified! They are derived from the position.
- The , separates the element-selections

- If more than one hierarchy is selected per dimension && splits the elementselections
- If no Hierarchy is specified. Default Hierarchy will be addressed

or Iterable [Element1, Element2, Element3] :param dimensions: optional. Dimension names in their natural order. Will speed up the execution! :param sandbox_name: str :param depth: optional. Depth of the component trace that will be returned. Deeper traces take longer :param element_separator: Alternative separator for the elements, if elements are passed as string :param hierarchy_separator: Alternative separator for multiple hierarchies, if elements are passed as string :param hierarchy_element_separator: Alternative separator between hierarchy name and element name, if elements are passed as string :return: trace json string

trace_cell_feeders(*cube_name: str*, *elements: Iterable | str*, *dimensions: Iterable[str] = None*, *sandbox_name: str = None*, *element_separator: str = ','*, *hierarchy_separator: str = '&&'*, *hierarchy_element_separator: str = '::'*, ***kwargs*) → Dict

Trace feeders from a cell

Parameters

- **cube_name** – name of the target cube
- **elements** –

string "Hierarchy1::Element1 && Hierarchy2::Element4, Element9, Element2"

- Dimensions are not specified! They are derived from the position.
- The , separates the element-selections
- If more than one hierarchy is selected per dimension && splits the elementselections
- If no Hierarchy is specified. Default Hierarchy will be addressed

or Iterable [Element1, Element2, Element3] :param dimensions: optional. Dimension names in their natural order. Will speed up the execution! :param sandbox_name: str :param element_separator: Alternative separator for the elements, if elements are passed as string :param hierarchy_separator: Alternative separator for multiple hierarchies, if elements are passed as string :param hierarchy_element_separator: Alternative separator between hierarchy name and element name, if elements are passed as string :return: feeder trace

transaction_log_is_active(*cube_name: str*) → bool

undo_changeset(*changeset: str*) → Response

undo a changeset. Similar to rolling back transactions.

Returns
Change set ID

update_cellset(*cellset_id: str*, *values: Iterable*, *sandbox_name: str = None*, *changeset: str = None*, ***kwargs*) → Response

Write values into cellset

Number of values must match the number of cells in the cellset

Parameters

- **cellset_id** –
- **values** – iterable with Numeric and String values
- **sandbox_name** – str

- **changeset** –

Returns

write(*cube_name: str*, *cellset_as_dict: Dict*, *dimensions: Iterable[str] = None*, *increment: bool = False*, *deactivate_transaction_log: bool = False*, *reactivate_transaction_log: bool = False*, *sandbox_name: str = None*, *use_ti: bool = False*, *use_blob: bool = False*, *use_changeset: bool = False*, *precision: int = None*, *skip_non_updateable: bool = False*, *measure_dimension_elements: Dict = None*, *remove_blob: bool = True*, *allow_spread: bool = False*, *clear_view: str = None*, ***kwargs*) → str | None

Write values to a cube

Same signature as *write_values* method, but faster since it uses *write_values_through_cellset* behind the scenes.

Supports incrementing cell values through optional *increment* argument Spreading through spreading shortcuts is not supported!

Parameters

- **cube_name** – name of the cube
- **cellset_as_dict** – {(elem_a, elem_b, elem_c): 243, (elem_d, elem_e, elem_f) : 109}
- **dimensions** – optional. Dimension names in their natural order. Will speed up the execution!
- **increment** – increment or update cell values
- **deactivate_transaction_log** – deactivate before writing
- **reactivate_transaction_log** – reactivate after writing
- **sandbox_name** – str
- **use_ti** – Use unbound process to write. Requires admin permissions. causes massive performance improvement.
- **use_blob** – Uses blob to write. Requires admin permissions. 10x faster compared to use_ti
- **use_changeset** – Enable ChangesetID: True or False
- **precision** – max precision when writhing through unbound process.

Necessary when dealing with large numbers to avoid "number too long" TI syntax error. :param skip_non_updateable skip cells that are not updateable (e.g. rule derived or consolidated) :param measure_dimension_elements: dictionary of measure elements and their types to improve performance when *use_ti* is *True*. When all written values are numeric you can pass a default dict with default key 'Numeric' :param remove_blob: remove blob file after writing with use_blob=True :param allow_spread: allow TI process in use_blob or use_ti to use CellPutProportionalSpread on C elements :param clear_view: name of cube view to clear before writing :return: changeset or None

write_async(*cube_name: str*, *cells: Dict*, *slice_size: int = 250000*, *max_workers: int = 8*, *dimensions: Iterable[str] = None*, *increment: bool = False*, *deactivate_transaction_log: bool = False*, *reactivate_transaction_log: bool = False*, *sandbox_name: str = None*, *precision: int = None*, *measure_dimension_elements: Dict = None*, ***kwargs*) → str | None

Write asynchronously

Parameters

- **cube_name** –
- **cells** –

- **slice_size** –
- **max_workers** –
- **dimensions** –
- **increment** –
- **deactivate_transaction_log** –
- **reactivate_transaction_log** –
- **sandbox_name** –
- **precision** – max precision when writhing through unbound process.

Necessary to decrease when dealing with large numbers to avoid "number too long" TI syntax error. :param measure_dimension_elements: dictionary of measure elements and their types to improve performance when *use_ti* is *True*. :param kwargs: :return:

write_dataframe(*cube_name: str*, *data: DataFrame*, *dimensions: Iterable[str] = None*, *increment: bool = False*, *deactivate_transaction_log: bool = False*, *reactivate_transaction_log: bool = False*, *sandbox_name: str = None*, *use_ti: bool = False*, *use_blob: bool = False*, *use_changeset: bool = False*, *precision: int = None*, *skip_non_updateable: bool = False*, *measure_dimension_elements: Dict = None*, *sum_numeric_duplicates: bool = True*, *remove_blob: bool = True*, *allow_spread: bool = False*, *clear_view: str = None*, ***kwargs*) → str

Function expects same shape as *execute_mdx_dataframe* returns. Column order must match dimensions in the target cube with an additional column for the values. Column names are not relevant. :param cube_name: :param data: Pandas Data Frame :param dimensions: :param increment: :param deactivate_transaction_log: :param reactivate_transaction_log: :param sandbox_name: :param use_ti: :param use_blob: Uses blob to write. Requires admin permissions. 10x faster compared to use_ti :param use_changeset: Enable ChangesetID: True or False :param precision: max precision when writhing through unbound process. Necessary when dealing with large numbers to avoid "number too long" TI syntax error :param skip_non_updateable skip cells that are not updateable (e.g. rule derived or consolidated) :param measure_dimension_elements: dictionary of measure elements and their types to improve performance when *use_ti* is *True*. When all written values are numeric you can pass a default dict with default key 'Numeric' :param sum_numeric_duplicates: Aggregate numerical values for duplicated intersections :param remove_blob: remove blob file after writing with use_blob=True :param allow_spread: allow TI process in use_blob or use_ti to use CellPutProportionalSpread on C elements :param clear_view: name of cube view to clear before writing :return: changeset or None

write_dataframe_async(*cube_name: str*, *data: DataFrame*, *slice_size_of_dataframe: int = 250000*, *max_workers: int = 8*, *dimensions: Iterable[str] = None*, *increment: bool = False*, *sandbox_name: str = None*, *deactivate_transaction_log: bool = False*, *reactivate_transaction_log: bool = False*, ***kwargs*)

Write DataFrame into a cube using unbound TI processes in a multi-threading way. Requires admin permissions. For a DataFrame with > 1,000,000 rows, this function will at least save half of runtime compared with *write_dataframe* function. Column order must match dimensions in the target cube with an additional column for the values. Column names are not relevant. :param cube_name: :param data: Pandas Data Frame :param slice_size_of_dataframe: Number of rows for each DataFrame slice, e.g. 10000 :param max_workers: Max number of threads, e.g. 14 :param dimensions: :param increment: increment or update cell values. Defaults to False. :param sandbox_name: name of the sandbox or None :param deactivate_transaction_log: :param reactivate_transaction_log: :return: the Future's result or raise exception.

write_through_blob(*cube_name: str*, *cellset_as_dict: dict*, *increment: bool = False*, *sandbox_name: str = None*, *skip_non_updateable: bool = False*, *remove_blob=True*, *dimensions: str = None*, *allow_spread: bool = False*, *clear_view: str = None*, ***kwargs*)

Writes data back to TM1 via an unbound TI process having an uploaded CSV as data source :param

cube_name: str :param cellset_as_dict: :param increment: increment or update cell values :param sandbox_name: str :param skip_non_updateable skip cells that are not updateable (e.g. rule derived or consolidated) :param remove_blob: choose False to persist blob after write. Can be helpful for troubleshooting. :param dimensions: optional. Dimension names in their natural order. Will speed up the execution! :param allow_spread: allow TI process in use_blob or use_ti to use CellPutProportionalSpread on C elements. :param clear_view: name of cube view to clear before writing :param kwargs: :return: Success: bool, Messages: list, ChangeSet: None

write_through_cellset(*cube_name: str*, *cellset_as_dict: Dict*, *dimensions: Iterable[str] = None*, *increment: bool = False*, *deactivate_transaction_log: bool = False*, *reactivate_transaction_log: bool = False*, *sandbox_name: str = None*, *use_changeset: bool = False*, *skip_non_updateable: bool = False*, ***kwargs*) → str

write_through_unbound_process(*cube_name: str*, *cellset_as_dict: Dict*, *increment: bool = False*, *sandbox_name: str = None*, *precision: int = None*, *skip_non_updateable: bool = False*, *measure_dimension_elements: Dict = None*, *is_attribute_cube: bool = None*, *dimensions: List = None*, *allow_spread: bool = False*, ***kwargs*)

Writes data back to TM1 via an unbound TI process :param cube_name: str :param cellset_as_dict: :param increment: increment or update cell values :param sandbox_name: str :param precision: max precision when writhing through unbound process. :param skip_non_updateable skip cells that are not updateable (e.g. rule derived or consolidated) :param measure_dimension_elements: pass dictionary of measure elements and their types to improve performance When all written values are numeric you can pass a defaultdict with default key: 'Numeric' :param is_attribute_cube bool or None :param allow_spread: allow TI process in use_blob or use_ti to use CellPutProportionalSpread on C elements :param kwargs: :return: Success: bool, Messages: list, ChangeSet: None

write_value(*value: str | float*, *cube_name: str*, *element_tuple: Iterable*, *dimensions: Iterable[str] = None*, *sandbox_name: str = None*, ***kwargs*) → Response

Write value into cube at specified coordinates

Parameters

- **value** – the actual value
- **cube_name** – name of the target cube
- **element_tuple** – target coordinates
- **dimensions** – optional. Dimension names in their natural order. Will speed up the execution!
- **sandbox_name** – str

Returns
response

write_values(*cube_name: str*, *cellset_as_dict: Dict*, *dimensions: Iterable[str] = None*, *sandbox_name: str = None*, *changeset: str = None*, ***kwargs*) → str

Write values to a cube

For cellsets with > 1000 cells look into *write* or *write_values_through_cellset* Supports spreading shortcuts

Parameters

- **cube_name** – name of the cube
- **cellset_as_dict** – {(elem_a, elem_b, elem_c): 243, (elem_d, elem_e, elem_f) : 109}

- **dimensions** – optional. Dimension names in their natural order. Will speed up the execution!
- **sandbox_name** – str
- **changeset** – str

Returns
Response

write_values_through_cellset(*mdx: str*, *values: Iterable*, *increment: bool = False*, *sandbox_name: str = None*, ***kwargs*) → str

Significantly faster than write_values function

Cellset gets created according to MDX Expression. For instance: [[61, 29 ,13], [42, 54, 15], [17, 28, 81]]

Each value in the cellset can be addressed through its position: The ordinal integer value. Ordinal-enumeration goes from top to bottom from left to right Number 61 has Ordinal 0, 29 has Ordinal 1, etc.

The order of the iterable determines the insertion point in the cellset. For instance: [91, 85, 72, 68, 51, 42, 35, 28, 11]

would lead to: [[91, 85 ,72], [68, 51, 42], [35, 28, 11]]

When writing large datasets into TM1 Cubes it can be convenient to call this function asynchronously.

Parameters

- **mdx** – Valid MDX Expression.
- **values** – List of values. The Order of the List/ Iterable determines the insertion point in the cellset.
- **increment** – increment or update cells
- **sandbox_name** – str

Returns
changeset: str

class TM1py.**ChoreService**(*rest:* RestService)

Service to handle Object Updates for TM1 Chores

activate(*chore_name: str*, ***kwargs*) → Response

activate chore on TM1 Server :param chore_name: :return: response

create(*chore:* Chore, ***kwargs*) → Response

create a chore :param chore: instance of TM1py.Chore :return:

deactivate(*chore_name: str*, ***kwargs*) → Response

deactivate chore on TM1 Server :param chore_name: :return: response

delete(*chore_name: str*, ***kwargs*) → Response

delete chore in TM1 :param chore_name: :return: response

execute_chore(*chore_name: str*, ***kwargs*) → Response

Ask TM1 Server to execute a chore :param chore_name: String, name of the chore to be executed :return: the response

exists(*chore_name: str*, ***kwargs*) → bool

Check if Chore exists

Parameters
chore_name –

Returns

get(*chore_name: str*, ***kwargs*) → *Chore*

Get a chore from the TM1 Server :param chore_name: :return: instance of TM1py.Chore

get_all(***kwargs*) → List[*Chore*]

get a List of all Chores :return: List of TM1py.Chore

get_all_names(***kwargs*) → List[str]

get a List of all Chores :return: List of TM1py.Chore

search_for_parameter_value(*parameter_value: str*, ***kwargs*) → List[*Chore*]

Return chore details for any/all chores that have a specified value set in the chore parameter settings
*this will NOT check the process parameter default, rather the defined parameter value saved in the chore

Parameters
parameter_value – string, will search wildcard for string in parameter value using Contains(string)

search_for_process_name(*process_name: str*, ***kwargs*) → List[*Chore*]

Return chore details for any/all chores that contain specified process name in tasks

Parameters
process_name – string, a valid ti process name; spaces will be elimniated

set_local_start_time(*chore_name: str*, *date_time: datetime*, ***kwargs*) → Response

Makes Server crash if chore is activated (10.2.2 FP6) :) :param chore_name: :param date_time: :return:

update(*chore:* Chore, ***kwargs*)

update chore on TM1 Server does not update: DST Sensitivity! :param chore: :return:

update_or_create(*chore:* Chore, ***kwargs*) → Response

static zfill_two(*number: int*) → str

Pad an int with zeros on the left two create two digit string

Parameters
number –

Returns

class TM1py.**CubeService**(*rest:* RestService)

Service to handle Object Updates for TM1 Cubes

check_rules(*cube_name: str*, ***kwargs*) → Response

Check rules syntax for existing cube on TM1 Server

Parameters
cube_name – name of a cube

Returns
response

create(*cube:* Cube, ***kwargs*) → Response

create new cube on TM1 Server

Parameters
cube – instance of TM1py.Cube

Returns
response

cube_save_data(*cube_name: str*, ***kwargs*) → Response

Serializes a cube by saving data updates

Parameters
cube_name –

Returns
Response

delete(*cube_name: str*, ***kwargs*) → Response

Delete a cube in TM1

Parameters
cube_name –

Returns
response

exists(*cube_name: str*, ***kwargs*) → bool

Check if a cube exists. Return boolean.

Parameters
cube_name –

Returns
Boolean

get(*cube_name: str*, ***kwargs*) → *Cube*

get cube from TM1 Server

Parameters
cube_name –

Returns
instance of TM1py.Cube

get_all(***kwargs*) → List[*Cube*]

get all cubes from TM1 Server as TM1py.Cube instances

Returns
List of TM1py.Cube instances

get_all_names(*skip_control_cubes: bool = False*, ***kwargs*) → List[str]

Ask TM1 Server for list of all cube names

Skip_control_cubes
bool, True will exclude control cubes from list

Returns
List of Strings

get_all_names_with_rules(*skip_control_cubes: bool = False*, ***kwargs*) → List[str]

Ask TM1 Server for list of all cube names that have rules

Skip_control_cubes
bool, True will exclude control cubes from list

Returns
List of Strings

get_all_names_without_rules(*skip_control_cubes: bool = False*, ***kwargs*) → List[str]

Ask TM1 Server for list of all cube names that do not have rules :skip_control_cubes: bool, True will exclude control cubes from list :return: List of Strings

get_control_cubes(***kwargs*) → List[*Cube*]

Get all Cubes with } prefix from TM1 Server as TM1py.Cube instances

Returns
List of TM1py.Cube instances

get_dimension_names(*cube_name: str*, *skip_sandbox_dimension: bool = True*, ***kwargs*) → List[str]

get name of the dimensions of a cube in their correct order

Parameters

- **cube_name** –
- **skip_sandbox_dimension** –

Returns
List : [dim1, dim2, dim3, etc.]

get_last_data_update(*cube_name: str*, ***kwargs*) → str

get_measure_dimension(*cube_name: str*, ***kwargs*) → str

get_model_cubes(***kwargs*) → List[*Cube*]

Get all Cubes without } prefix from TM1 Server as TM1py.Cube instances

Returns
List of TM1py.Cube instances

get_number_of_cubes(*skip_control_cubes: bool = False*, ***kwargs*) → int

Ask TM1 Server for count of cubes

Skip_control_cubes
bool, True will exclude control cubes from count

Returns
int, count

get_random_intersection(*cube_name: str*, *unique_names: bool = False*) → List[str]

Get a random Intersection in a cube used mostly for regression testing. Not optimized, in terms of performance. Function Loads ALL elements for EACH dim…

Parameters

- **cube_name** –
- **unique_names** – unique names instead of plain element names

Returns
List of elements

get_storage_dimension_order(*cube_name: str*, ***kwargs*) → List[str]

Get the storage dimension order of a cube

Parameters
cube_name –

Returns
List of dimension names

load(*cube_name: str*, ***kwargs*) → Response

Load the cube into memory on the server

Parameters
cube_name –

Returns

lock(*cube_name: str*, ***kwargs*) → Response

Locks the cube to prevent any users from modifying it

Parameters
cube_name –

Returns

search_for_dimension(*dimension_name: str*, *skip_control_cubes: bool = False*, ***kwargs*) → List[str]

Ask TM1 Server for list of cube names that contain specific dimension

Parameters

- **dimension_name** – string, valid dimension name (case insensitive)
- **skip_control_cubes** – bool, True will exclude control cubes from result

search_for_dimension_substring(*substring: str*, *skip_control_cubes: bool = False*, ***kwargs*) → Dict[str, List[str]]

Ask TM1 Server for a dictinary of cube names with the dimension whose name contains the substring

Parameters

- **substring** – string to search for in dim name
- **skip_control_cubes** – bool, True will exclude control cubes from result

search_for_rule_substring(*substring: str*, *skip_control_cubes: bool = False*, *case_insensitive=True*, *space_insensitive=True*, ***kwargs*) → List[*Cube*]

get all cubes from TM1 Server as TM1py.Cube instances where rules for given cube contain specified substring

Parameters

- **substring** – string to search for in rules
- **skip_control_cubes** – bool, True will exclude control cubes from result
- **case_insensitive** – case agnostic search
- **space_insensitive** – space agnostic search

Returns
List of TM1py.Cube instances

unload(*cube_name: str*, ***kwargs*) → Response

Unload the cube from memory

Parameters
cube_name –

Returns

unlock(*cube_name: str*, ***kwargs*) → Response

Unlocks the cube to allow modifications

Parameters
cube_name –

Returns

update(*cube:* Cube, ***kwargs*) → Response

Update existing cube on TM1 Server

Parameters
cube – instance of TM1py.Cube

Returns
response

update_or_create(*cube:* Cube, ***kwargs*) → Response

update if exists else create

Parameters
cube –

Returns

update_storage_dimension_order(*cube_name: str*, *dimension_names: Iterable[str]*) → float

Update the storage dimension order of a cube

Parameters

- **cube_name** –
- **dimension_names** –

Returns
Float: -23.076489699337078 (percent change in memory usage)

class TM1py.**DimensionService**(*rest:* RestService)

Service to handle Object Updates for TM1 Dimensions

create(*dimension:* Dimension, ***kwargs*) → Response

Create a dimension

Parameters
dimension – instance of TM1py.Dimension

Returns
response

create_element_attributes_through_ti(*dimension:* Dimension, ***kwargs*)

:param dimension. Instance of TM1py.Objects.Dimension class :return:

delete(*dimension_name: str*, ***kwargs*) → Response

Delete a dimension

Parameters
dimension_name – Name of the dimension

Returns

execute_mdx(*dimension_name: str*, *mdx: str*, ***kwargs*) → List

Execute MDX against Dimension. Requires }ElementAttributes_ Cube of the dimension to exist !

Parameters

- **dimension_name** – Name of the Dimension
- **mdx** – valid Dimension-MDX Statement

Returns
List of Element names

exists(*dimension_name: str*, ***kwargs*) → bool

Check if dimension exists

Returns

get(*dimension_name: str*, ***kwargs*) → *Dimension*

Get a Dimension

Parameters
dimension_name –

Returns

get_all_names(*skip_control_dims: bool = False*, ***kwargs*) → List[str]

Ask TM1 Server for list of all dimension names

Skip_control_dims
bool, True to skip control dims

Returns
List of Strings

get_number_of_dimensions(*skip_control_dims: bool = False*, ***kwargs*) → int

Ask TM1 Server for number of dimensions

Skip_control_dims
bool, True to exclude control dims from count

Returns
Number of dimensions

update(*dimension:* Dimension, *keep_existing_attributes=False*, ***kwargs*)

Update an existing dimension

Parameters

- **dimension** – instance of TM1py.Dimension
- **keep_existing_attributes** – True to make sure existing attributes are not removed

Returns
None

update_or_create(*dimension:* Dimension, ***kwargs*)

update if exists else create

Parameters
dimension –

Returns

uses_alternate_hierarchies(*dimension_name: str*, ***kwargs*) → bool

class TM1py.**ElementService**(*rest:* RestService)

Service to handle Object Updates for TM1 Dimension (resp. Hierarchy) Elements

add_edges(*dimension_name: str*, *hierarchy_name: str = None*, *edges: Dict[Tuple[str, str], int] = None*, ***kwargs*) → Response

Add Edges to hierarchy. Fails if one edge already exists.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **edges** –

Returns

add_element_attributes(*dimension_name: str*, *hierarchy_name: str*, *element_attributes: List[*ElementAttribute*]*, ***kwargs*)

Add element attributes to hierarchy. Fails if one element attribute already exists.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **element_attributes** –

Returns

add_elements(*dimension_name: str*, *hierarchy_name: str*, *elements: Iterable[*Element*]*, ***kwargs*)

Add elements to hierarchy. Fails if one element already exists.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **elements** –

Returns

attribute_cube_exists(*dimension_name: str*, ***kwargs*) → bool

create(*dimension_name: str*, *hierarchy_name: str*, *element:* Element, ***kwargs*) → Response

create_element_attribute(*dimension_name: str*, *hierarchy_name: str*, *element_attribute:* ElementAttribute, ***kwargs*) → Response

like AttrInsert

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **element_attribute** – instance of TM1py.ElementAttribute

Returns

delete(*dimension_name: str*, *hierarchy_name: str*, *element_name: str*, ***kwargs*) → Response

delete_edges(*dimension_name: str*, *hierarchy_name: str*, *edges: Iterable[Tuple[str, str]] = None*, *use_ti: bool = False*, ***kwargs*)

delete_edges_use_ti(*dimension_name: str*, *hierarchy_name: str*, *edges: List[str] = None*, ***kwargs*)

delete_element_attribute(*dimension_name: str*, *hierarchy_name: str*, *element_attribute: str*, ***kwargs*) → Response

like AttrDelete

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **element_attribute** – instance of TM1py.ElementAttribute

Returns

delete_elements(*dimension_name: str*, *hierarchy_name: str*, *element_names: List[str] = None*, *use_ti: bool = False*, ***kwargs*)

delete_elements_use_ti(*dimension_name: str*, *hierarchy_name: str*, *element_names: List[str] = None*, ***kwargs*)

element_is_ancestor(*dimension_name: str*, *hierarchy_name: str*, *ancestor_name: str*, *element_name: str*, *method: str = None*) → bool

Element is Ancestor

:Note, unlike the related function in TM1 (*ELISANC* or *ElementIsAncestor*), this function will return False if an invalid element is passed; but will raise an exception if an invalid dimension, or hierarchy is passed

For *method* you can pass 3 three values value *TI* performs best, but requires admin permissions Value 'TM1DrillDownMember' performs well when element is a leaf. Value 'Descendants' performs well when *ancestor_name* and *element_name* are Consolidations.

If no value is passed, function defaults to 'TI' for user with admin permissions and 'TM1DrillDownMember' for users without admin permissions

element_is_parent(*dimension_name: str*, *hierarchy_name: str*, *parent_name: str*, *element_name: str*) → bool

Element is Parent :Note, unlike the related function in TM1 (ELISPAR or ElementIsParent), this function will return False :if an invalid element is passed; :but will raise an exception if an invalid dimension, or hierarchy is passed

execute_set_mdx(*mdx: str*, *top_records: int | None = None*, *member_properties: Iterable[str] | None = ('Name', 'Weight')*, *parent_properties: Iterable[str] | None = ('Name', 'UniqueName')*, *element_properties: Iterable[str] | None = ('Type', 'Level')*, ***kwargs*) → List

:method to execute an MDX statement against a dimension :param mdx: valid dimension mdx statement :param top_records: number of records to return, default: all elements no limit :param member_properties: list of member properties (e.g., Name, UniqueName, Type, Weight, Attributes/Color) to return, will always return the Name property :param parent_properties: list of parent properties (e.g., Name, UniqueName, Type, Weight, Attributes/Color)

to return, can be None or empty

Parameters

element_properties – list of element properties (e.g., Name, UniqueName, Type, Level, Index,

Attributes/Color) to return, can be empty :return: dictionary of members, unique names, weights, types, and parents

exists(*dimension_name: str*, *hierarchy_name: str*, *element_name: str*, ***kwargs*) → bool

get(*dimension_name: str*, *hierarchy_name: str*, *element_name: str*, ***kwargs*) → *Element*

get_alias_element_attributes(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[str]

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns

get_all_element_identifiers(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → CaseAndSpaceInsensitiveSet

Get all element names and alias values in a hierarchy

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns

get_all_leaf_element_identifiers(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → CaseAndSpaceInsensitiveSet

Get all element names and alias values for leaf elements in a hierarchy

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns

get_attribute_of_elements(*dimension_name: str*, *hierarchy_name: str*, *attribute: str*, *elements: str | List[str] = None*, *exclude_empty_cells: bool = True*, *element_unique_names: bool = False*) → dict

Get element name and attribute value for a set of elements in a hierarchy

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **attribute** – Name of the Attribute
- **elements** – MDX (Set) expression or iterable of elements
- **exclude_empty_cells** – Boolean
- **element_unique_names** – Boolean

Returns

Dict {'01':'Jan', '02':'Feb'}

get_consolidated_element_names(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[str]

get_consolidated_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[*Element*]

get_edges(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → Dict[Tuple[str, str], int]

get_edges_under_consolidation(*dimension_name: str*, *hierarchy_name: str*, *consolidation: str*, *max_depth: int = None*, ***kwargs*) → List[str]

Get all members under a consolidated element

Parameters

- **dimension_name** – name of dimension
- **hierarchy_name** – name of hierarchy
- **consolidation** – name of consolidated Element
- **max_depth** – 99 if not passed

Returns

get_element_attribute_names(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[str]

Get element attributes from hierarchy

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns

get_element_attributes(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[*ElementAttribute*]

Get element attributes from hierarchy

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns

get_element_identifiers(*dimension_name: str*, *hierarchy_name: str*, *elements: str | List[str]*, ***kwargs*) → CaseAndSpaceInsensitiveSet

Get all element names and alias values for a set of elements in a hierarchy

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **elements** – MDX (Set) expression or iterable of elements

Returns

get_element_names(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[str]

Get all element names

Parameters

- **dimension_name** –

- **hierarchy_name** –

Returns
Generator of element-names

get_element_principal_name(*dimension_name: str*, *hierarchy_name: str*, *element_name: str*, ***kwargs*) → str

get_element_types(*dimension_name: str*, *hierarchy_name: str*, *skip_consolidations: bool = False*, ***kwargs*) → CaseAndSpaceInsensitiveDict

get_element_types_from_all_hierarchies(*dimension_name: str*, *skip_consolidations: bool = False*, ***kwargs*) → CaseAndSpaceInsensitiveDict

get_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[*Element*]

get_elements_by_level(*dimension_name: str*, *hierarchy_name: str*, *level: int*, ***kwargs*) → List[str]

Get all element names by level in a hierarchy

Parameters

- **dimension_name** – Name of the dimension
- **hierarchy_name** – Name of the hierarchy
- **level** – Level to filter

Returns
List of element names

get_elements_dataframe(*dimension_name: str = None*, *hierarchy_name: str = None*, *elements: str | Iterable[str] = None*, *skip_consolidations: bool = True*, *attributes: Iterable[str] = None*, *attribute_column_prefix: str = ''*, *skip_parents: bool = False*, *level_names: List[str] = None*, *parent_attribute: str = None*, *skip_weights: bool = False*, *use_blob: bool = False*, *allow_empty_alias: bool = True*, ***kwargs*) → DataFrame

Parameters

- **dimension_name** – Name of the dimension. Can be derived from elements MDX
- **hierarchy_name** – Name of the hierarchy in the dimension.Can be derived from elements MDX
- **elements** – Selection of members. Iterable or valid MDX string
- **skip_consolidations** – Boolean flag to skip consolidations
- **attributes** – Selection of attributes. Iterable. If None retrieve all.
- **attribute_column_prefix** – string to prefix attribute colums to avoid name conflicts
- **level_names** – List of labels for parent columns. If None use level names from TM1.
- **skip_parents** – Boolean Flag to skip parent columns.
- **parent_attribute** – Attribute to be displayed in parent columns. If None, parent name is used.
- **skip_weights** – include weight columns
- **use_blob** – Up to 40% better performance and lower memory footprint in any case. Requires admin permissions

- **allow_empty_alias** – False if empty alias values should be substituted with element names instead

Returns
pandas DataFrame

get_elements_filtered_by_attribute(*dimension_name: str*, *hierarchy_name: str*, *attribute_name: str*, *attribute_value: str | float*, ***kwargs*) → List[str]

Get all elements from a hierarchy with given attribute value

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **attribute_name** –
- **attribute_value** –

Returns
List of element names

get_elements_filtered_by_wildcard(*dimension_name: str*, *hierarchy_name: str*, *wildcard: str*, *level: int = None*, ***kwargs*) → List[str]

Get all element names filtered by wildcard (CaseAndSpaceInsensitive) and level in a hierarchy

Parameters

- **dimension_name** – Name of the dimension
- **hierarchy_name** – Name of the hierarchy
- **wildcard** – wildcard to filter
- **level** – Level to filter

Returns
List of element names

get_leaf_element_names(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[str]

get_leaf_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[*Element*]

get_leaves_under_consolidation(*dimension_name: str*, *hierarchy_name: str*, *consolidation: str*, *max_depth: int = None*, ***kwargs*) → List[str]

Get all leaves under a consolidated element

Parameters

- **dimension_name** – name of dimension
- **hierarchy_name** – name of hierarchy
- **consolidation** – name of consolidated Element
- **max_depth** – 99 if not passed

Returns

get_level_names(*dimension_name: str*, *hierarchy_name: str*, *descending: bool = True*, ***kwargs*) → List[str]

get_levels_count(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → int

get_members_under_consolidation(*dimension_name: str*, *hierarchy_name: str*, *consolidation: str*, *max_depth: int = None*, *leaves_only: bool = False*, ***kwargs*) → List[str]

Get all members under a consolidated element

Parameters

- **dimension_name** – name of dimension
- **hierarchy_name** – name of hierarchy
- **consolidation** – name of consolidated Element
- **max_depth** – 99 if not passed
- **leaves_only** – Only Leaf Elements or all Elements

Returns

get_number_of_consolidated_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → int

get_number_of_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → int

get_number_of_leaf_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → int

get_number_of_numeric_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → int

get_number_of_string_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → int

get_numeric_element_names(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[str]

get_numeric_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[*Element*]

get_parents(*dimension_name: str*, *hierarchy_name: str*, *element_name: str*, ***kwargs*) → List[str]

get_parents_of_all_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → Dict[str, List[str]]

get_process_service()

get_string_element_names(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[str]

get_string_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[*Element*]

hierarchy_exists(*dimension_name*, *hierarchy_name*)

remove_edge(*dimension_name: str*, *hierarchy_name: str*, *parent: str*, *component: str*, ***kwargs*) → Response

Remove one edge from hierarchy. Fails if parent or child element doesn't exist.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **parent** –
- **component** –

Returns

update(*dimension_name: str*, *hierarchy_name: str*, *element:* Element, ***kwargs*) → Response

update_or_create(*dimension_name: str*, *hierarchy_name: str*, *element:* Element, ***kwargs*) → Response

class TM1py.**FileService**(*tm1_rest: <module 'TM1py.Services.RestService' from '/home/docs/checkouts/readthedocs.org/user_builds/tm1py/checkouts/latest/TM1py/Services/RestService*

create(*file_name: str*, *file_content: bytes*, ***kwargs*)

delete(*file_name: str*, ***kwargs*)

exists(*file_name: str*, ***kwargs*)

get(*file_name: str*, ***kwargs*) → bytes

get_names(***kwargs*) → bytes

update(*file_name: str*, *file_content: bytes*, ***kwargs*)

update_or_create(*file_name: str*, *file_content: bytes*, ***kwargs*)

class TM1py.**GitService**(*rest:* RestService)

Service to interact with GIT

COMMON_PARAMETERS = {'author': 'Author', 'branch': 'Branch', 'config': 'Config', 'email': 'Email', 'force': 'Force', 'message': 'Message', 'new_branch': 'NewBranch', 'passphrase': 'Passphrase', 'password': 'Password', 'private_key': 'PrivateKey', 'public_key': 'PublicKey', 'username': 'Username'}

git_execute_plan(*plan_id: str*, ***kwargs*) → Response

Executes a plan based on the planid :param plan_id: GitPlan id

git_get_plans(***kwargs*) → List[*GitPlan*]

Gets a list of currently available GIT plans

git_init(*git_url: str*, *deployment: str*, *username: str = None*, *password: str = None*, *public_key: str = None*, *private_key: str = None*, *passphrase: str = None*, *force: bool = None*, *config: dict = None*, ***kwargs*) → *Git*

Initialize GIT service, returns Git object :param git_url: file or http(s) path to GIT repository :param deployment: name of selected deployment group :param username: GIT username :param password: GIT password :param public_key: SSH public key, available from PAA V2.0.9.4 :param private_key: SSH private key, available from PAA V2.0.9.4 :param passphrase: Passphrase for decrypting private key, if set :param force: reset git context on True :param config: Dictionary containing git configuration parameters

git_pull(*branch: str*, *force: bool = None*, *execute: bool = None*, *username: str = None*, *password: str = None*, *public_key: str = None*, *private_key: str = None*, *passphrase: str = None*, ***kwargs*) → Response

Creates a gitpull plan, returns response :param branch: The name of source branch :param force: A flag passed in for evaluating preconditions :param execute: Executes the plan right away if True :param username: GIT username :param password: GIT password :param public_key: SSH public key, available from PAA V2.0.9.4 :param private_key: SSH private key, available from PAA V2.0.9.4 :param passphrase: Passphrase for decrypting private key, if set

git_push(*message: str*, *author: str*, *email: str*, *branch: str = None*, *new_branch: str = None*, *force: bool = False*, *username: str = None*, *password: str = None*, *public_key: str = None*, *private_key: str = None*, *passphrase: str = None*, *execute: bool = None*, ***kwargs*) → Response

Creates a gitpush plan, returns response :param message: Commit message :param author: Name of commit author :param email: Email of commit author :param branch: The branch which last commit will be used as parent commit for new branch. Must be empty if GIT repo is empty :param new_branch: If specified,

creates a new branch and pushes the commit onto it. If not specified, pushes to the branch specified in "Branch" :param force: A flag passed in for evaluating preconditions :param username: GIT username :param password: GIT password :param public_key: SSH public key, available from PAA V2.0.9.4 :param private_key: SSH private key, available from PAA V2.0.9.4 :param passphrase: Passphrase for decrypting private key, if set :param execute: Executes the plan right away if True

git_status(*username: str = None*, *password: str = None*, *public_key: str = None*, *private_key: str = None*, *passphrase: str = None*, ***kwargs*) → *Git*

Get GIT status, returns Git object :param username: GIT username :param password: GIT password :param public_key: SSH public key, available from PAA V2.0.9.4 :param private_key: SSH private key, available from PAA V2.0.9.4 :param passphrase: Passphrase for decrypting private key, if set

git_uninit(*force: bool = False*, ***kwargs*)

Unitialize GIT service

Parameters
force – clean up git context when True

tm1project_delete()

tm1project_get() → TM1Project

summary

tm1project_put(*tm1_project: TM1Project*) → TM1Project

class TM1py.**HierarchyService**(*rest:* RestService)

Service to handle Object Updates for TM1 Hierarchies

EDGES_WORKAROUND_VERSIONS = ('11.0.002', '11.0.003', '11.1.000')

add_edges(*dimension_name: str*, *hierarchy_name: str = None*, *edges: Dict[Tuple[str, str], int] = None*, ***kwargs*) → Response

Add Edges to hierarchy. Fails if one edge already exists.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **edges** –

Returns

add_element_attributes(*dimension_name: str*, *hierarchy_name: str*, *element_attributes: List[*ElementAttribute*]*, ***kwargs*)

Add element attributes to hierarchy. Fails if one element attribute already exists.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **element_attributes** –

Returns

add_elements(*dimension_name: str*, *hierarchy_name: str*, *elements: List[*Element*]*, ***kwargs*)

Add elements to hierarchy. Fails if one element already exists.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **elements** –

Returns

create(*hierarchy: Hierarchy*, ***kwargs*)

Create a hierarchy in an existing dimension

Parameters
hierarchy –

Returns

delete(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → Response

exists(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → bool

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns

get(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → *Hierarchy*

get hierarchy

Parameters

- **dimension_name** – name of the dimension
- **hierarchy_name** – name of the hierarchy

Returns

get_all_names(*dimension_name: str*, ***kwargs*) → List[str]

get all names of existing Hierarchies in a dimension

Parameters
dimension_name –

Returns

get_cell_service()

get_default_member(*dimension_name: str*, *hierarchy_name: str = None*, ***kwargs*) → str | None

Get the defined default_member for a Hierarchy. Will return the element with index 1, if default member is not specified explicitly in }HierarchyProperty Cube

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns
String, name of Member

get_dimension_service()

get_hierarchy_summary(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → Dict[str, int]

is_balanced(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*)

Check if hierarchy is balanced

Parameters

- **dimension_name** –
- **hierarchy_name** –

Returns

remove_all_edges(*dimension_name: str*, *hierarchy_name: str = None*, ***kwargs*) → Response

remove_edges_under_consolidation(*dimension_name: str*, *hierarchy_name: str*, *consolidation_element: str*, ***kwargs*) → List[Response]

Parameters

- **dimension_name** – Name of the dimension
- **hierarchy_name** – Name of the hierarchy
- **consolidation_element** – Name of the Consolidated element

Returns
response

update(*hierarchy: Hierarchy*, *keep_existing_attributes=False*, ***kwargs*) → List[Response]

update a hierarchy. It's a two step process: 1. Update Hierarchy 2. Update Element-Attributes

Function caters for Bug with Edge Creation: https://www.ibm.com/developerworks/community/forums/html/topic?id=75f2b99e-6961-4c71-9364-1d5e1e083eff

Parameters

- **hierarchy** – instance of TM1py.Hierarchy
- **keep_existing_attributes** – True to make sure existing attributes are not removed

Returns
list of responses

update_default_member(*dimension_name: str*, *hierarchy_name: str = None*, *member_name: str = ''*, ***kwargs*) → Response

Update the default member of a hierarchy.

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **member_name** –

Returns

update_element_attributes(*hierarchy: Hierarchy*, *keep_existing_attributes=False*, ***kwargs*)

Update the elementattributes of a hierarchy

Parameters

- **hierarchy** – Instance of TM1py.Hierarchy
- **keep_existing_attributes** – True to make sure existing attributes are not removed

Returns

update_or_create(*hierarchy: Hierarchy*, ***kwargs*)

update if exists else create

Parameters
hierarchy –

Returns

update_or_create_hierarchy_from_dataframe(*dimension_name: str*, *hierarchy_name: str*, *df: DataFrame*, *element_column: str = None*, *verify_unique_elements: bool = False*, *verify_edges: bool = True*, *element_type_column: str = 'ElementType'*, *unwind: bool = False*)

Update or Create a hierarchy based on a dataframe, while never deleting existing elements.

Parameters

- **dimension_name** – Name of the dimension
- **hierarchy_name** – Name of the hierarchy
- **df** – pd.DataFrame the data frame. Example: | | Region | ElementType | Alias:a | Currency:s | population:n | level001 | level000 | level001_weight | level000_weight | |---:|:———|:------------|:————|:-----------|————-:|:---------|:———|----------------:|—————-:| | 0 | France | Numeric | Frankreich | EUR | 60000000 | Europe | World | 1 | 1 | | 1 | Belgium | Numeric | Schweiz | CHF | 9000000 | Europe | World | 1 | 1 | | 2 | Germany | Numeric | Deutschland | EUR | 84000000 | Europe | World | 1 | 1 |

  Names for the parent columns (level001, level000) are not configurable and *level000* is the top node. All columns except for the element_column, element_type_colums and parent columns are attribute columns. On attribute columns, you specify the type as a suffix. If no type is provided string attributes are created
- **element_type_column** – str The column name in the df which specifies which element is which type. If None, all will be considered N level.
- **element_column** – str The column name of the element ID. If None, assumes first column is the element ID.
- **verify_unique_elements** – Abort early if element names are not unique
- **verify_edges** – Abort early if edges have circular reference
- **unwind** – bool Unwind hierarch before creating new edges

Returns

class TM1py.**MonitoringService**(*rest: RestService*)

Service to Query and Cancel Threads in TM1

cancel_all_running_threads(***kwargs*) → list

cancel_thread(*thread_id: int*, ***kwargs*) → Response

Kill a running thread

Parameters
thread_id –

Returns

close_all_sessions(***kwargs*) → list

close_session(*session_id*, ***kwargs*) → Response

disconnect_all_users(***kwargs*) → list

disconnect_user(*user_name: str*, ***kwargs*) → Response

Disconnect User

Parameters
user_name –

Returns

get_active_session_threads(*exclude_idle: bool = True*, ***kwargs*)

get_active_threads(***kwargs*)

Return a list of non-idle threads from the TM1 Server

Returns
list: TM1 threads as dict

get_active_users(***kwargs*) → List[*User*]

Get the activate users in TM1

Returns
List of TM1py.User instances

get_current_user(***kwargs*)

get_sessions(*include_user: bool = True*, *include_threads: bool = True*, ***kwargs*) → List

get_threads(***kwargs*) → List

Return a dict of the currently running threads from the TM1 Server

Returns
dict: the response

user_is_active(*user_name: str*, ***kwargs*) → bool

Check if user is currently active in TM1

Parameters
user_name –

Returns
Boolean

class TM1py.**PowerBiService**(*tm1_rest*)

execute_mdx(*mdx*, ***kwargs*) → DataFrame

execute_view(*cube_name: str*, *view_name: str*, *private: bool*, *use_iterative_json=False*, *use_blob=False*, ***kwargs*) → DataFrame

get_member_properties(*dimension_name: str = None*, *hierarchy_name: str = None*, *member_selection: Iterable = None*, *skip_consolidations: bool = True*, *attributes: Iterable = None*, *skip_parents: bool = False*, *level_names=None*, *parent_attribute: str = None*, *skip_weights=True*, *use_blob=False*, ***kwargs*) → DataFrame

Parameters

- **dimension_name** – Name of the dimension
- **hierarchy_name** – Name of the hierarchy in the dimension

- **member_selection** – Selection of members. Iterable or valid MDX string
- **skip_consolidations** – Boolean flag to skip consolidations
- **attributes** – Selection of attributes. Iterable. If None retrieve all.
- **level_names** – List of labels for parent columns. If None use level names from TM1.
- **skip_parents** – Boolean Flag to skip parent columns.
- **parent_attribute** – Attribute to be displayed in parent columns. If None, parent name is used.
- **skip_weights** – include weight columns
- **use_blob** – Better performance on large sets and lower memory footprint in any case. Requires admin permissions

Returns
pandas DataFrame

class TM1py.**ProcessService**(*rest:* RestService)

Service to handle Object Updates for TI Processes

compile(*name: str*, ***kwargs*) → List

Compile a Process. Return List of Syntax errors.

Parameters
name –

Returns

compile_process(*process:* Process, ***kwargs*) → List

Compile a Process. Return List of Syntax errors.

Parameters
process –

Returns

create(*process:* Process, ***kwargs*) → Response

Create a new process on TM1 Server

Parameters
process – Instance of TM1py.Process class

Returns
Response

debug_add_breakpoint(*debug_id: str*, *break_point: ProcessDebugBreakpoint*, ***kwargs*) → Response

debug_add_breakpoints(*debug_id: str*, *break_points: Iterable[ProcessDebugBreakpoint] = None*, ***kwargs*) → Response

debug_continue(*debug_id: str*, ***kwargs*) → Dict

Resumes execution until next breakpoint

debug_get_breakpoints(*debug_id: str*, ***kwargs*) → List[ProcessDebugBreakpoint]

debug_get_current_breakpoint(*debug_id: str*, ***kwargs*) → ProcessDebugBreakpoint

debug_get_process_line_number(*debug_id: str*, ***kwargs*) → str

debug_get_process_procedure(*debug_id: str*, ***kwargs*) → str

debug_get_record_number(*debug_id: str*, ***kwargs*) → str

debug_get_single_variable_value(*debug_id: str*, *variable_name: str*, ***kwargs*) → str

debug_get_variable_values(*debug_id: str*, ***kwargs*) → CaseInsensitiveDict

debug_process(*process_name: str*, *timeout: float = None*, ***kwargs*) → Dict

Start debug session for specified process; debug session id is returned in response

debug_remove_breakpoint(*debug_id: str*, *breakpoint_id: int*, ***kwargs*) → Response

debug_step_in(*debug_id: str*, ***kwargs*) → Dict

Runs a single statement in the process If ExecuteProcess is next function, will pause at first statement inside child process

debug_step_out(*debug_id: str*, ***kwargs*) → Dict

Resumes execution and runs until current process has finished.

debug_step_over(*debug_id: str*, ***kwargs*) → Dict

Runs a single statement in the process If ExecuteProcess is next function, will NOT debug child process

debug_update_breakpoint(*debug_id: str*, *break_point: ProcessDebugBreakpoint*, ***kwargs*) → Response

delete(*name: str*, ***kwargs*) → Response

Delete a process in TM1

Parameters
name –

Returns
Response

evaluate_boolean_ti_expression(*formula: str*)

evaluate_ti_expression(*formula: str*, ***kwargs*) → str

This function is same functionality as hitting "Evaluate" within variable formula editor in TI
Function creates temporary TI and then starts a debug session on that TI EnableTIDebugging=T must be present in .cfg file Only suited for DEV and one-off uses, don't incorporate into dataframe lambda function

Parameters
formula – a valid tm1 variable formula (no double quotes, no equals sign, semicolon optional) e.g. "8*2;", "CellGetN('c1', 'e1', 'e2);", "ATTRS('Region', 'France', 'Currency')"

Returns
string result from formula

execute(*process_name: str*, *parameters: Dict = None*, *timeout: float = None*, *cancel_at_timeout: bool = False*, ***kwargs*) → Response

Ask TM1 Server to execute a process. Call with parameter names as keyword arguments: tm1.processes.execute("Bedrock.Server.Wait", pLegalEntity="UK01")

Parameters

- **process_name** –

- **parameters** – Deprecated! dictionary, e.g. {"Parameters": [{ "Name": "pLegalEntity", "Value": "UK01" }] }
- **timeout** – Number of seconds that the client will wait to receive the first byte.
- **cancel_at_timeout** – Abort operation in TM1 when timeout is reached

Returns

execute_process_with_return(*process: Process*, *timeout: float = None*, *cancel_at_timeout: bool = False*, ***kwargs*) → Tuple[bool, str, str]

Run unbound TI code directly.

Parameters

- **process** – a TI Process Object
- **timeout** – Number of seconds that the client will wait to receive the first byte.
- **cancel_at_timeout** – Abort operation in TM1 when timeout is reached
- **kwargs** – dictionary of process parameters and values

Returns

success (boolean), status (String), error_log_file (String)

execute_ti_code(*lines_prolog: Iterable[str]*, *lines_epilog: Iterable[str] = None*, ***kwargs*) → Response

Execute lines of code on the TM1 Server

Parameters

- **lines_prolog** – list - where each element is a valid statement of TI code.
- **lines_epilog** – list - where each element is a valid statement of TI code.

execute_with_return(*process_name: str = None*, *timeout: float = None*, *cancel_at_timeout: bool = False*, *return_async_id: bool = False*, ***kwargs*) → Tuple[bool, str, str]

Ask TM1 Server to execute a process. pass process parameters as keyword arguments to this function. E.g:

self.tm1.processes.execute_with_return(
process_name="Bedrock.Server.Wait", pWaitSec=2)

Parameters

- **process_name** – name of the TI process
- **timeout** – Number of seconds that the client will wait to receive the first byte.
- **cancel_at_timeout** – Abort operation in TM1 when timeout is reached
- **return_async_id** – return async_id instead of (success, status, error_log_file)
- **kwargs** – dictionary of process parameters and values

Returns

success (boolean), status (String), error_log_file (String)

exists(*name: str*, ***kwargs*) → bool

Check if Process exists.

Parameters
name –

Returns

get(*name_process: str*, ***kwargs*) → *Process*

Get a process from TM1 Server

Parameters
name_process –

Returns
Instance of the TM1py.Process

get_all(*skip_control_processes: bool = False*, ***kwargs*) → List[*Process*]

Get a processes from TM1 Server

Parameters
skip_control_processes – bool, True to exclude processes that begin with "}" or "{"

Returns
List, instances of the TM1py.Process

get_all_names(*skip_control_processes: bool = False*, ***kwargs*) → List[str]

Get List with all process names from TM1 Server

Parameters
skip_control_processes – bool, True to exclude processes that begin with "}" or "{"

Returns
List of Strings

get_error_log_file_content(*file_name: str*, ***kwargs*) → str

Get content of error log file (e.g. TM1ProcessError_20180926213819_65708356_979b248b-232e622c6.log)

Parameters
file_name – name of the error log file in the TM1 log directory

Returns
String, content of the file

get_error_log_filenames(*process_name: str = None*, *top: int = 0*, *descending: bool = False*, ***kwargs*) → List[str]

Get error log filenames for specified TI process

Parameters

- **process_name** – valid TI name, leave blank to return all error log filenames
- **top** – top n filenames
- **descending** – default sort is ascending, descending=True would have most recent at the top of list

Returns
list of filenames

get_last_message_from_processerrorlog(*process_name: str*, ***kwargs*) → str

Get the latest ProcessErrorLog from a process entity

Parameters
process_name – name of the process

Returns
String - the errorlog, e.g.: "Error: Data procedure line (9): Invalid key:

Dimension Name: "Product", Element Name (Key): "ProductA""

get_processerrorlogs(*process_name: str*, ***kwargs*) → List

Get all ProcessErrorLog entries for a process

Parameters
process_name – name of the process

Returns
list - Collection of ProcessErrorLogs

poll_execute_with_return(*async_id: str*)

search_error_log_filenames(*search_string: str*, *top: int = 0*, *descending: bool = False*, ***kwargs*) → List[str]

Search error log filenames for given search string like a datestamp e.g. 20231201

Parameters

- **search_string** – substring to contain in file names
- **top** – top n filenames
- **descending** – default sort is ascending, descending=True would have most recent at the top of list

Returns
list of filenames

search_string_in_code(*search_string: str*, *skip_control_processes: bool = False*, ***kwargs*) → List[str]

Ask TM1 Server for list of process names that contain string anywhere in code tabs: Prolog,Metadata,Data,Epilog will not search DataSource, Parameters, Variables, or Attributes

Parameters

- **search_string** – case insensitive string to search for
- **skip_control_processes** – bool, True to exclude processes that begin with "}" or "{"

Returns
List of strings

search_string_in_name(*name_startswith: str = None*, *name_contains: Iterable = None*, *name_contains_operator: str = 'and'*, *skip_control_processes: bool = False*, ***kwargs*) → List[str]

Ask TM1 Server for list of process names that contain or start with string

Parameters

- **name_startswith** – str, process name begins with (case insensitive)
- **name_contains** – iterable, found anywhere in name (case insensitive)
- **name_contains_operator** – 'and' or 'or'
- **skip_control_processes** – bool, True to exclude processes that begin with "}" or "{"

update(*process: Process*, ***kwargs*) → Response

Update an existing Process on TM1 Server

Parameters
process – Instance of TM1py.Process class

Returns
Response

update_or_create(*process:* Process, ***kwargs*) → Response

Update or Create a Process on TM1 Server

Parameters
process – Instance of TM1py.Process class

Returns
Response

class TM1py.**RestService**(***kwargs*)

Low level communication with TM1 instance through HTTP. Allows to execute HTTP Methods

- GET
- POST
- PATCH
- DELETE

Takes Care of

- Encodings
- TM1 User-Login
- HTTP Headers
- HTTP Session Management
- Response Handling

Based on requests module

DEFAULT_CONNECTION_POOL_SIZE = 10

DELETE(*url: str*, *data: str | bytes = ''*, *headers: Dict = None*, *async_requests_mode: bool = None*, *return_async_id: bool = False*, *timeout: float = None*, *cancel_at_timeout: bool = False*, *encoding: str = 'utf-8'*, ***kwargs*)

Perform a GET request against TM1 instance :param url: :param data: the payload :param headers: custom headers :param async_requests_mode changes internal REST execution mode to avoid 60s timeout on IBM cloud :param return_async_id: If True function will return async_id after initiation and not await the execution :param timeout: Number of seconds that the client will wait to receive the first byte. :param cancel_at_timeout: Abort operation in TM1 when timeout is reached :param encoding: :return: response object or async_id

GET(*url: str*, *data: str | bytes = ''*, *headers: Dict = None*, *async_requests_mode: bool = None*, *return_async_id: bool = False*, *timeout: float = None*, *cancel_at_timeout: bool = False*, *encoding: str = 'utf-8'*, ***kwargs*)

Perform a GET request against TM1 instance :param url: :param data: the payload :param headers: custom headers :param async_requests_mode changes internal REST execution mode to avoid 60s timeout on IBM cloud :param return_async_id: If True function will return async_id after initiation and not await the execution :param timeout: Number of seconds that the client will wait to receive the first byte. :param cancel_at_timeout: Abort operation in TM1 when timeout is reached :param encoding: :return: response object or async_id

HEADERS = {'Accept': 'application/json;odata.metadata=none,text/plain', 'Connection': 'keep-alive', 'Content-Type': 'application/json; odata.streaming=true; charset=utf-8', 'TM1-SessionContext': 'TM1py', 'User-Agent': 'TM1py'}

PATCH(*url: str*, *data: str | bytes = ''*, *headers: Dict = None*, *async_requests_mode: bool = None*, *return_async_id: bool = False*, *timeout: float = None*, *cancel_at_timeout: bool = False*, *encoding: str = 'utf-8'*, ***kwargs*)

Perform a GET request against TM1 instance :param url: :param data: the payload :param headers: custom headers :param async_requests_mode changes internal REST execution mode to avoid 60s timeout on IBM cloud :param return_async_id: If True function will return async_id after initiation and not await the execution :param timeout: Number of seconds that the client will wait to receive the first byte. :param cancel_at_timeout: Abort operation in TM1 when timeout is reached :param encoding: :return: response object or async_id

POST(*url: str*, *data: str | bytes = ''*, *headers: Dict = None*, *async_requests_mode: bool = None*, *return_async_id: bool = False*, *timeout: float = None*, *cancel_at_timeout: bool = False*, *encoding: str = 'utf-8'*, ***kwargs*)

Perform a GET request against TM1 instance :param url: :param data: the payload :param headers: custom headers :param async_requests_mode changes internal REST execution mode to avoid 60s timeout on IBM cloud :param return_async_id: If True function will return async_id after initiation and not await the execution :param timeout: Number of seconds that the client will wait to receive the first byte. :param cancel_at_timeout: Abort operation in TM1 when timeout is reached :param encoding: :return: response object or async_id

PUT(*url: str*, *data: str | bytes = ''*, *headers: Dict = None*, *async_requests_mode: bool = None*, *return_async_id: bool = False*, *timeout: float = None*, *cancel_at_timeout: bool = False*, *encoding: str = 'utf-8'*, ***kwargs*)

Perform a GET request against TM1 instance :param url: :param data: the payload :param headers: custom headers :param async_requests_mode changes internal REST execution mode to avoid 60s timeout on IBM cloud :param return_async_id: If True function will return async_id after initiation and not await the execution :param timeout: Number of seconds that the client will wait to receive the first byte. :param cancel_at_timeout: Abort operation in TM1 when timeout is reached :param encoding: :return: response object or async_id

TCP_SOCKET_OPTIONS = {'TCP_KEEPCNT': 60, 'TCP_KEEPIDLE': 30, 'TCP_KEEPINTVL': 15}

add_compact_json_header() → str

add_http_header(*key: str*, *value: str*)

static b64_decode_password(*encrypted_password: str*) → str

b64 decoding :param encrypted_password: encrypted password with b64 :return: password in plain text

static build_response_from_binary_response(*data: bytes*) → Response

cancel_async_operation(*async_id: str*, ***kwargs*)

cancel_running_operation()

connect()

static disable_http_warnings()

get_api_metadata() → dict

Get API Metadata

Returns
Dictionary

get_http_header(*key: str*) → str

get_monitoring_service()

handle_logging(*logging: str | bool*)

property is_admin: bool

is_connected() → bool

Check if Connection to TM1 Server is established. :Returns:

Boolean

property is_data_admin: bool

property is_ops_admin: bool

property is_security_admin: bool

logout(*timeout: float = None*, ***kwargs*)

End TM1 Session and HTTP session

remove_http_header(*key: str*)

request(*method: str*, *url: str*, *data: str = ''*, *encoding='utf-8'*, *async_requests_mode: bool | None = None*, *return_async_id=False*, *timeout: float = None*, *cancel_at_timeout: bool = False*, ***kwargs*)

retrieve_async_response(*async_id: str*, ***kwargs*) → Response

property sandboxing_disabled

property session_id: str

set_version()

static translate_to_boolean(*value*) → bool

Takes a boolean or string (eg. true, True, FALSE, etc.) value and returns (boolean) True or False :param value: True, 'true', 'false' or 'False' … :return:

static urllib3_response_from_bytes(*data: bytes*) → HTTPResponse

Build urllib3.HTTPResponse based on raw bytes string

static verify_response(*response: Response*)

check if Status Code is OK :Parameters:

***response*: String**
the response that is returned from a method call

Exceptions
TM1pyException, raises TM1pyException when Code is not 200, 204 etc.

property version: str

static wait_time_generator(*timeout: int*)

class TM1py.**SandboxService**(*rest:* RestService)

Service to handle sandboxes in TM1

create(*sandbox:* Sandbox, ***kwargs*) → Response

create a new sandbox in TM1 Server

Parameters
sandbox – Sandbox

Returns
response

delete(*sandbox_name: str*, ***kwargs*) → Response

delete a sandbox in TM1

Parameters
sandbox_name –

Returns
response

exists(*sandbox_name: str*, ***kwargs*) → bool

check if the sandbox exists in TM1

Parameters
sandbox_name – String

Returns
bool

get(*sandbox_name: str*, ***kwargs*) → *Sandbox*

get a sandbox from TM1 Server

Parameters
sandbox_name – str

Returns
instance of TM1py.Sandbox

get_all(***kwargs*) → List[*Sandbox*]

get all sandboxes from TM1 Server

Returns
List of TM1py.Sandbox instances

get_all_names(***kwargs*) → List[str]

get all sandbox names

Parameters
kwargs –

Returns

load(*sandbox_name: str*, ***kwargs*) → Response

load sandbox into memory

Parameters
sandbox_name – str

Returns
response

merge(*source_sandbox_name: str*, *target_sandbox_name: str*, *clean_after: bool = False*, ***kwargs*) → Response

merge one sandbox into another

Parameters

- **source_sandbox_name** – str
- **target_sandbox_name** – str
- **clean_after** – bool: Reset source sandbox after merging

Returns
response

publish(*sandbox_name: str*, ***kwargs*) → Response

publish existing sandbox to base

Parameters
sandbox_name – str

Returns
response

reset(*sandbox_name: str*, ***kwargs*) → Response

reset all changes in specified sandbox

Parameters
sandbox_name – str

Returns
response

unload(*sandbox_name: str*, ***kwargs*) → Response

unload sandbox from memory

Parameters
sandbox_name – str

Returns
response

update(*sandbox:* Sandbox, ***kwargs*) → Response

update a sandbox in TM1

Parameters
sandbox –

Returns
response

class TM1py.**SecurityService**(*rest:* RestService)

Service to handle Security stuff

add_user_to_groups(*user_name: str*, *groups: Iterable[str]*, ***kwargs*) → Response

Parameters

- **user_name** – name of user
- **groups** – iterable of groups

Returns
response

create_group(*group_name: str*, ***kwargs*) → Response

Create a Security group in the TM1 Server

Parameters
group_name –

Returns

create_user(*user:* User, ***kwargs*) → Response

Create a user on TM1 Server

Parameters
user – instance of TM1py.User

Returns
response

delete_group(*group_name: str*, ***kwargs*) → Response

Delete a group in the TM1 Server

Parameters
group_name –

Returns

delete_user(*user_name: str*, ***kwargs*) → Response

Delete user on TM1 Server

Parameters
user_name –

Returns
response

determine_actual_group_name(*group_name: str*, ***kwargs*) → str

determine_actual_user_name(*user_name: str*, ***kwargs*) → str

get_all_groups(***kwargs*) → List[str]

Get all groups from TM1 Server

Returns
List of strings

get_all_user_names(***kwargs*)

Get all user names from TM1 Server

Returns
List of TM1py.User instances

get_all_users(***kwargs*)

Get all users from TM1 Server

Returns
List of TM1py.User instances

get_current_user(***kwargs*) → *User*

Get user and group assignments of this session

Returns
instance of TM1py.User

get_custom_security_groups(***kwargs*) → List[str]

get_groups(*user_name: str*, ***kwargs*) → List[str]

Get the groups of a user in TM1 Server

Parameters
user_name –

Returns
List of strings

get_read_only_users(***kwargs*) → List[str]

get_user(*user_name: str*, ***kwargs*) → *User*

Get user from TM1 Server

Parameters
user_name –

Returns
instance of TM1py.User

get_user_names_from_group(*group_name: str*, ***kwargs*) → List[str]

Get all users from group

Parameters
group_name –

Returns
List of strings

get_users_from_group(*group_name: str*, ***kwargs*)

Get all users from group

Parameters
group_name –

Returns
List of TM1py.User instances

group_exists(*group_name: str*, ***kwargs*) → bool

remove_user_from_group(*group_name: str*, *user_name: str*, ***kwargs*) → Response

Remove user from group in TM1 Server

Parameters

- **group_name** –
- **user_name** –

Returns
response

security_refresh(***kwargs*) → Response

update_user(*user:* User, ***kwargs*) → Response

Update user on TM1 Server

Parameters
user – instance of TM1py.User

Returns
response

update_user_password(*user_name: str*, *password: str*, ***kwargs*) → Response

user_exists(*user_name: str*, ***kwargs*) → bool

class TM1py.**ServerService**(*rest:* RestService)

Service to query common information from the TM1 Server

activate_audit_log()

deactivate_audit_log()

delete_persistent_feeders(***kwargs*) → Response

execute_audit_log_delta_request(***kwargs*) → Dict

execute_message_log_delta_request(***kwargs*) → Dict

execute_transaction_log_delta_request(***kwargs*) → Dict

get_active_configuration(***kwargs*) → Dict

Read effective(!) TM1 config settings as dictionary from TM1 Server

Returns
config as dictionary

get_admin_host(***kwargs*) → str

get_all_message_logger_level()

Get tm1 message log levels :param logger: :param level: :return:

get_api_metadata()

Read effective(!) TM1 config settings as dictionary from TM1 Server

Returns
config as dictionary

get_audit_log_entries(*user: str = None*, *object_type: str = None*, *object_name: str = None*, *since: datetime = None*, *until: datetime = None*, *top: int = None*, ***kwargs*) → Dict

Parameters

- **user** – UserName
- **object_type** – ObjectType
- **object_name** – ObjectName
- **since** – of type datetime. If it doesn't have tz information, UTC is assumed.
- **until** – of type datetime. If it doesn't have tz information, UTC is assumed.
- **top** – int

Returns

get_configuration(***kwargs*) → Dict

get_data_directory(***kwargs*) → str

get_last_process_message_from_message_log(*process_name: str*, ***kwargs*) → str | None

Get the latest message log entry for a process

Parameters

process_name – name of the process

Returns

String - the message, for instance: "Ausführung normal beendet, verstrichene Zeit 0.03 Sekunden"

get_message_log_entries(*reverse: bool = True*, *since: datetime = None*, *until: datetime = None*, *top: int = None*, *logger: str = None*, *level: str = None*, *msg_contains: Iterable = None*, *msg_contains_operator: str = 'and'*, ***kwargs*) → Dict

Parameters

- **reverse** – Boolean
- **since** – of type datetime. If it doesn't have tz information, UTC is assumed.
- **until** – of type datetime. If it doesn't have tz information, UTC is assumed.
- **top** – Integer
- **logger** – string, eg TM1.Server, TM1.Chore, TM1.Mdx.Interface, TM1.Process
- **level** – string, ERROR, WARNING, INFO, DEBUG, UNKNOWN
- **msg_contains** – iterable, find substring in log message; list of substrings will be queried as AND statement
- **msg_contains_operator** – 'and' or 'or'
- **kwargs** –

Returns

Dict of server log

get_product_version(***kwargs*) → str

Ask TM1 Server for its version

Returns

String, the version

get_server_name(***kwargs*) → str

Ask TM1 Server for its name

Returns

String, the server name

get_static_configuration(***kwargs*) → Dict

get_transaction_log_entries(*reverse: bool = True*, *user: str = None*, *cube: str = None*, *since: datetime = None*, *until: datetime = None*, *top: int = None*, *element_tuple_filter: Dict[str, str] = None*, *element_position_filter: Dict[int, Dict[str, str]] = None*, ***kwargs*) → Dict

Parameters

- **reverse** – Boolean
- **user** – UserName
- **cube** – CubeName

- **since** – of type datetime. If it doesn't have tz information, UTC is assumed.
- **until** – of type datetime. If it doesn't have tz information, UTC is assumed.
- **top** – int
- **element_tuple_filter** – of type dict. Element name as key and comparison operator as value
- **element_position_filter** – not yet implemented

tuple={'Actual':'eq','2020': 'ge'} :return:

initialize_audit_log_delta_requests(*filter=None*, ***kwargs*)

initialize_message_log_delta_requests(*filter=None*, ***kwargs*)

initialize_transaction_log_delta_requests(*filter=None*, ***kwargs*)

save_data(***kwargs*) → Response

start_performance_monitor()

stop_performance_monitor()

update_message_logger_level(*logger*, *level*)

Updates tm1 message log levels :param logger: :param level: :return:

update_static_configuration(*configuration: Dict*) → Response

Update the .cfg file and triggers TM1 to re-read the file.

Parameters
configuration –

Returns
Response

static utc_localize_time(*timestamp*)

write_to_message_log(*level: str*, *message: str*, ***kwargs*) → None

Parameters

- **level** – string, FATAL, ERROR, WARN, INFO, DEBUG
- **message** – string

Returns

class TM1py.**SubsetService**(*rest:* RestService)

Service to handle Object Updates for TM1 Subsets (dynamic and static)

create(*subset:* Subset, *private: bool = False*, ***kwargs*) → Response

create subset on the TM1 Server

Parameters

- **subset** – TM1py.Subset, the subset that shall be created
- **private** – boolean

Returns
string: the response

delete(*subset_name: str*, *dimension_name: str*, *hierarchy_name: str = None*, *private: bool = False*, ***kwargs*) → Response

Delete an existing subset on the TM1 Server

Parameters

- **subset_name** – String, name of the subset
- **dimension_name** – String, name of the dimension
- **hierarchy_name** – String, name of the hierarchy
- **private** – Boolean

Returns

delete_elements_from_static_subset(*dimension_name: str*, *hierarchy_name: str*, *subset_name: str*, *private: bool*, ***kwargs*) → Response

exists(*subset_name: str*, *dimension_name: str*, *hierarchy_name: str = None*, *private: bool = False*, ***kwargs*) → bool

checks if private or public subset exists

Parameters

- **subset_name** –
- **dimension_name** –
- **hierarchy_name** –
- **private** –

Returns
boolean

get(*subset_name: str*, *dimension_name: str*, *hierarchy_name: str = None*, *private: bool = False*, ***kwargs*) → *Subset*

get a subset from the TM1 Server

Parameters

- **subset_name** – string, name of the subset
- **dimension_name** – string, name of the dimension
- **hierarchy_name** – string, name of the hierarchy
- **private** – Boolean

Returns
instance of TM1py.Subset

get_all_names(*dimension_name: str*, *hierarchy_name: str = None*, *private: bool = False*, ***kwargs*) → List[str]

get names of all private or public subsets in a hierarchy

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **private** – Boolean

Returns
List of Strings

get_element_names(*dimension_name: str*, *hierarchy_name: str*, *subset_name: str*, *private: bool = False*, ***kwargs*)

Get elements from existing (dynamic or static) subset

Parameters

- **dimension_name** –
- **hierarchy_name** –
- **subset_name** –
- **private** –
- **kwargs** –

Returns

make_static(*subset_name: str*, *dimension_name: str*, *hierarchy_name: str = None*, *private: bool = False*) → Response

convert a dynamic subset into static subset on the TM1 Server :param subset_name: String, name of the subset :param dimension_name: String, name of the dimension :param hierarchy_name: String, name of the hierarchy :param private: Boolean :return: response

update(*subset:* Subset, *private: bool = False*, ***kwargs*) → Response

update a subset on the TM1 Server

Parameters

- **subset** – instance of TM1py.Subset.
- **private** – Boolean

Returns
response

update_or_create(*subset:* Subset, *private: bool = False*, ***kwargs*) → Response

update if exists else create

Parameters

- **subset** –
- **private** –

Returns

class TM1py.**TM1Service**(***kwargs*)

All features of TM1py are exposed through this service

Can be saved and restored from File, to avoid multiple authentication with TM1.

property connection

logout(***kwargs*)

property metadata

re_authenticate()

re_connect()

classmethod restore_from_file(*file_name*)

save_to_file(*file_name*)

property version

property whoami

class TM1py.**ViewService**(*rest:* RestService)

Service to handle Object Updates for cube views (NativeViews and MDXViews)

create(*view:* MDXView | NativeView, *private: bool = False*, ***kwargs*) → Response

create a new view on TM1 Server

Parameters

- **view** – instance of subclass of TM1py.View (TM1py.NativeView or TM1py.MDXView)
- **private** – boolean

Returns
Response

delete(*cube_name: str*, *view_name: str*, *private: bool = False*, ***kwargs*) → Response

Delete an existing view (MDXView or NativeView) on the TM1 Server

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the view
- **private** – Boolean

Returns
String, the response

exists(*cube_name: str*, *view_name: str*, *private: bool = None*, ***kwargs*)

Checks if view exists as private, public or both

Parameters

- **cube_name** – string, name of the cube
- **view_name** – string, name of the view
- **private** – boolean, if None: check for private and public

:return boolean tuple

get(*cube_name: str*, *view_name: str*, *private: bool = False*, ***kwargs*) → *View*

get_all(*cube_name: str*, *include_elements: bool = True*, ***kwargs*) → Tuple[List[*View*], List[*View*]]

Get all public and private views from cube. :param cube_name: String, name of the cube. :param include_elements: false to return view details without elements, faster :return: 2 Lists of TM1py.View instances: private views, public views

get_all_names(*cube_name: str*, ***kwargs*) → Tuple[List[str], List[str]]

Parameters
cube_name –

Returns

get_mdx_view(*cube_name: str*, *view_name: str*, *private: bool = False*, ***kwargs*) → *MDXView*

Get an MDXView from TM1 Server

Parameters

- **cube_name** – String, name of the cube
- **view_name** – String, name of the MDX view
- **private** – boolean

Returns

instance of TM1py.MDXView

get_native_view(*cube_name: str*, *view_name: str*, *private=False*, ***kwargs*) → *NativeView*

Get a NativeView from TM1 Server

Parameters

- **cube_name** – string, name of the cube
- **view_name** – string, name of the native view
- **private** – boolean

Returns

instance of TM1py.NativeView

is_mdx_view(*cube_name: str*, *view_name: str*, *private=False*, ***kwargs*)

is_native_view(*cube_name: str*, *view_name: str*, *private=False*)

search_subset_in_native_views(*dimension_name: str = None*, *subset_name: str = None*, *cube_name: str = None*, *include_elements: bool = False*, ***kwargs*) → Tuple[List[*View*], List[*View*]]

Get all public and private native views that utilize specified dimension subset

Parameters

- **dimension_name** – string, valid dimension name with subset to query
- **subset_name** – string, valid subset name to search for in views
- **cube_name** – str, optionally specify cube to search, otherwise will search all cubes
- **include_elements** – false to return view details without elements, faster

Returns

2 Lists of TM1py.View instances: private views, public views

update(*view: MDXView | NativeView*, *private: bool = False*, ***kwargs*) → Response

Update an existing view

Parameters

- **view** – instance of TM1py.NativeView or TM1py.MDXView
- **private** – boolean

Returns

response

update_or_create(*view: MDXView | NativeView*, *private: bool = False*, ***kwargs*) → Response

update if exists, else create

Parameters

- **view** –
- **private** –
- **kwargs** –

Returns

TM1 Objects

class TM1py.**Annotation**(*comment_value: str*, *object_name: str*, *dimensional_context: Iterable[str]*, *comment_type: str = 'ANNOTATION'*, *annotation_id: str = None*, *text: str = ''*, *creator: str = None*, *created: str = None*, *last_updated_by: str = None*, *last_updated: str = None*)

Abtraction of TM1 Annotation

Notes

- Class complete, functional and tested.
- doesn't cover Attachments though

property body: str

property body_as_dict: Dict

property comment_value: str

construct_body_for_post(*cube_dimensions*) → Dict

property created: str

property dimensional_context: List[str]

classmethod from_json(*annotation_as_json: str*) → *Annotation*

Alternative constructor

Parameters
annotation_as_json – String, JSON

Returns
instance of TM1py.Process

property id: str

property last_updated: str

property last_updated_by: str

move(*dimension_order: Iterable[str]*, *dimension: str*, *target_element: str*, *source_element: str = None*)

Move annotation on given dimension from source_element to target_element

Parameters

- **dimension_order** – List, order of the dimensions in the cube
- **dimension** – dimension name

- **target_element** – target element name
- **source_element** – source element name

Returns

property object_name: str

property text: str

class TM1py.**Application**(*path: str*, *name: str*, *application_type: ApplicationTypes | str*)

property application_id: str

property body: str

property body_as_dict: Dict

class TM1py.**Chore**(*name: str*, *start_time:* ChoreStartTime, *dst_sensitivity: bool*, *active: bool*, *execution_mode: str*, *frequency:* ChoreFrequency, *tasks: Iterable[*ChoreTask*]*)

Abstraction of TM1 Chore

MULTIPLE_COMMIT = 'MultipleCommit'

SINGLE_COMMIT = 'SingleCommit'

activate()

property active: bool

add_task(*task:* ChoreTask)

property body: str

property body_as_dict: Dict

construct_body() → str

construct self.body (json) from the class attributes :return: String, TM1 JSON representation of a chore

deactivate()

property dst_sensitivity: bool

property execution_mode: str

property execution_path: Dict

1 chore together with its executed processes Use case: building out a tree of chores and their processes (and again the processes that are called by the latter (if any)). :return: dictionary containing chore name as the key and a list of process names as the value

property frequency: *ChoreFrequency*

classmethod from_dict(*chore_as_dict: Dict*) → *Chore*

Alternative constructor

Parameters
chore_as_dict – Chore as dict

Returns
Chore, an instance of this class

classmethod from_json(*chore_as_json: str*) → *Chore*

Alternative constructor

Parameters
chore_as_json – string, JSON. Response of /Chores('x')/Tasks?$expand=*

Returns
Chore, an instance of this class

property name: str

reschedule(*days: int = 0*, *hours: int = 0*, *minutes: int = 0*, *seconds: int = 0*)

property start_time: *ChoreStartTime*

property tasks: List[*ChoreTask*]

class TM1py.**ChoreFrequency**(*days: str | int*, *hours: str | int*, *minutes: str | int*, *seconds: str | int*)

Utility class to handle time representation fore Chore Frequency

property days: str

property frequency_string: str

classmethod from_string(*frequency_string: str*) → *ChoreFrequency*

property hours: str

property minutes: str

property seconds: str

class TM1py.**ChoreStartTime**(*year: int*, *month: int*, *day: int*, *hour: int*, *minute: int*, *second: int*, *tz: str = None*)

Utility class to handle time representation for Chore Start Time

add(*days: int = 0*, *hours: int = 0*, *minutes: int = 0*, *seconds: int = 0*)

property datetime: <module 'datetime' from '/home/docs/.asdf/installs/python/3.11.6/lib/python3.11/datetime.py'>

classmethod from_string(*start_time_string: str*) → *ChoreStartTime*

set_time(*year: int = None*, *month: int = None*, *day: int = None*, *hour: int = None*, *minute: int = None*, *second: int = None*)

property start_time_string: str

subtract(*days: int = 0*, *hours: int = 0*, *minutes: int = 0*, *seconds: int = 0*)

class TM1py.**ChoreTask**(*step: int*, *process_name: str*, *parameters: List[Dict[str, str]]*)

Abstraction of a Chore Task

A Chore task always conistst of - The step integer ID: it's order in the execution plan.

1 to n, where n is the last Process in the Chore

- The name of the process to execute
- The parameters for the process

property body: `str`

property body_as_dict: `Dict`

classmethod from_dict(*chore_task_as_dict: Dict*, *step: int = None*)

property parameters: `List[Dict[str, str]]`

property process_name: `str`

property step: `int`

class TM1py.**Cube**(*name: str*, *dimensions: Iterable[str]*, *rules: str | Rules | None = None*)

Abstraction of a TM1 Cube

property body: `str`

property dimensions: `List[str]`

property feedstrings: `bool`

classmethod from_dict(*cube_as_dict: Dict*) → *Cube*

Alternative constructor

Parameters
cube_as_dict – user as dict

Returns
user, an instance of this class

classmethod from_json(*cube_as_json: str*) → *Cube*

Alternative constructor

Parameters
cube_as_json – user as JSON string

Returns
cube, an instance of this class

property has_rules: `bool`

property name: `str`

property rules: *Rules*

property skipcheck: `bool`

property undefvals: `bool`

class TM1py.**Dimension**(*name: str*, *hierarchies: Iterable[Hierarchy] | None = None*)

Abstraction of TM1 Dimension

A Dimension is a container for hierarchies.

add_hierarchy(*hierarchy: Hierarchy*)

property body: `str`

property body_as_dict: `Dict`

contains_hierarchy(*hierarchy_name: str*) → bool

property default_hierarchy: *Hierarchy*

classmethod from_dict(*dimension_as_dict: Dict*) → *Dimension*

classmethod from_json(*dimension_as_json: str*) → *Dimension*

get_hierarchy(*hierarchy_name: str*) → *Hierarchy*

property hierarchies: List[*Hierarchy*]

property hierarchy_names: List[str]

property name: str

remove_hierarchy(*hierarchy_name: str*)

property unique_name: str

class TM1py.Element(*name*, *element_type:* Types | *str*, *attributes: List[str] = None*, *unique_name: str = None*, *index: int = None*)

Abstraction of TM1 Element

ELEMENT_ATTRIBUTES_PREFIX = '}ElementAttributes_'

class Types(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

CONSOLIDATED = 3

NUMERIC = 1

STRING = 2

property body: str

property body_as_dict: Dict

property element_attributes: List[str]

property element_type: *Types*

static from_dict(*element_as_dict: Dict*) → *Element*

property index: int

property name: str

property unique_name: str

class TM1py.ElementAttribute(*name: str*, *attribute_type:* Types | *str*)

Abstraction of TM1 Element Attributes

class Types(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

ALIAS = 3

NUMERIC = 1

STRING = 2

property attribute_type: str

property body: str

property body_as_dict: Dict

classmethod from_dict(*element_attribute_as_dict: Dict*) → *ElementAttribute*

classmethod from_json(*element_attribute_as_json: str*) → *ElementAttribute*

property name: str

class TM1py.**Git**(*url: str*, *deployment: str*, *force: bool*, *deployed_commit:* GitCommit, *remote:* GitRemote, *config: dict = None*)

Abstraction of Git object

property config: dict

property deployed_commit: *GitCommit*

property deployment: str

property force: bool

classmethod from_dict(*json_response: Dict*) → *Git*

property remote: *GitRemote*

property url: str

class TM1py.**GitCommit**(*commit_id: str*, *summary: str*, *author: str*)

Abstraction of Git Commit

property author: str

property commit_id: str

property summary: str

class TM1py.**GitPlan**(*plan_id: str*, *branch: str*, *force: bool*)

Base GitPlan abstraction

property branch: str

property force: bool

property plan_id: str

class TM1py.**GitRemote**(*connected: bool*, *branches: List[str]*, *tags: List[str]*)

Abstraction of GitRemote

property branches: List[str]

property connected: bool

property tags: List[str]

class TM1py.**Hierarchy**(*name: str*, *dimension_name: str*, *elements: Iterable[Element] | None = None*, *element_attributes: Iterable[ElementAttribute] | None = None*, *edges: Dict | None = None*, *subsets: Iterable[str] | None = None*, *structure: int | None = None*, *default_member: str | None = None*)

Abstraction of TM1 Hierarchy Requires reference to a Dimension

Elements modeled as a Dictionary where key is the element name and value an instance of TM1py.Element {

'US': instance of TM1py.Element, 'CN': instance of TM1py.Element, 'AU': instance of TM1py.Element

}

ElementAttributes of type TM1py.Objects.ElementAttribute

Edges are represented as a TM1py.Utils.CaseAndSpaceInsensitiveTupleDict: {

(parent1, component1) : 10, (parent1, component2) : 30

}

Subsets is list of type TM1py.Subset

add_component(*parent_name: str*, *component_name: str*, *weight: int*)

add_edge(*parent: str*, *component: str*, *weight: float*)

add_element(*element_name: str*, *element_type: str | Types*)

add_element_attribute(*name: str*, *attribute_type: str*)

property balanced: bool

property body: str

property body_as_dict: Dict

contains_element(*element_name: str*) → bool

property default_member: str

property dimension_name: str

property edges: Dict[Tuple[str], *Element*]

property element_attributes: List[*ElementAttribute*]

property elements: Dict[str, *Element*]

classmethod from_dict(*hierarchy_as_dict: Dict*, *dimension_name: str = None*) → *Hierarchy*

get_ancestor_edges(*element_name: str*, *recursive: bool = False*) → Dict

get_ancestors(*element_name: str*, *recursive: bool = False*) → Set[*Element*]

get_descendant_edges(*element_name: str*, *recursive: bool = False*) → Dict

get_descendants(*element_name: str*, *recursive: bool = False*, *leaves_only=False*) → Set[*Element*]

get_element(*element_name: str*) → *Element*

property name: str

remove_all_edges()

remove_all_elements()

remove_edge(*parent: str*, *component: str*)

remove_edges(*edges: Iterable[Tuple[str, str]]*)

remove_edges_related_to_element(*element_name: str*)

remove_element(*element_name: str*)

remove_element_attribute(*name: str*)

replace_element(*old_element_name: str*, *new_element_name: str*)

Substitute one element in the hierarchy structure, so that all edges are moved from the old element to the new element.

property subsets: List[str]

update_edge(*parent: str*, *component: str*, *weight: float*)

update_element(*element_name: str*, *element_type: str | Types*)

class TM1py.**MDXView**(*cube_name: str*, *view_name: str*, *MDX: str*)

Abstraction on TM1 MDX view

IMPORTANT. MDXViews can't be seen through the old TM1 clients (Archict, Perspectives). They do exist though!

property MDX: str

property body: str

construct_body() → str

classmethod from_dict(*view_as_dict: Dict*, *cube_name: str = None*) → *MDXView*

classmethod from_json(*view_as_json: str*, *cube_name: str | None = None*) → *MDXView*

property mdx

substitute_title(*dimension: str*, *hierarchy: str*, *element: str*)

dimension and hierarchy name are space sensitive!

Parameters

- **dimension** –
- **hierarchy** –
- **element** –

Returns

class TM1py.**NativeView**(*cube_name: str*, *view_name: str*, *suppress_empty_columns: bool | None = False*, *suppress_empty_rows: bool | None = False*, *format_string: str | None = '0.#########'*, *titles: Iterable[ViewTitleSelection] | None = None*, *columns: Iterable[ViewAxisSelection] | None = None*, *rows: Iterable[ViewAxisSelection] | None = None*)

Abstraction of TM1 NativeView (classic cube view)

Notes

Complete, functional and tested

property MDX: str

add_column(*dimension_name: str*, *subset:* Subset | *AnonymousSubset = None*)

Add Dimension or Subset to the column-axis

Parameters

- **dimension_name** – name of the dimension
- **subset** – instance of TM1py.Subset. Can be None

Returns

add_row(*dimension_name: str*, *subset:* Subset *= None*)

Add Dimension or Subset to the row-axis

Parameters

- **dimension_name** –
- **subset** – instance of TM1py.Subset. Can be None instead.

Returns

add_title(*dimension_name: str*, *selection: str*, *subset:* Subset | *AnonymousSubset = None*)

Add subset and element to the titles-axis

Parameters

- **dimension_name** – name of the dimension.
- **selection** – name of an element.
- **subset** – instance of TM1py.Subset. Can be None instead.

Returns

property as_MDX: str

Build a valid MDX Query from an Existing cubeview. Takes Zero suppression into account. Throws an Exception when no elements are place on the columns. Subsets are referenced in the result-MDX through the TM1SubsetToSet Function

Returns

String, the MDX Query

property body: str

property columns: List[*ViewAxisSelection*]

property format_string: str

classmethod from_dict(*view_as_dict: Dict*, *cube_name: str = None*) → *NativeView*

classmethod from_json(*view_as_json: str*, *cube_name: str | None = None*) → *NativeView*

Alternative constructor :Parameters:

view_as_json : string, JSON

Returns

View : an instance of this class

property mdx

remove_column(*dimension_name: str*)

remove dimension from the column axis

Parameters
dimension_name –

Returns

remove_row(*dimension_name: str*)

remove dimension from the row axis

Parameters
dimension_name –

Returns

remove_title(*dimension_name: str*)

Remove dimension from the titles-axis

Parameters
dimension_name – name of the dimension.

Returns

property rows: List[*ViewAxisSelection*]

substitute_title(*dimension: str*, *element: str*)

property suppress_empty_cells: bool

property suppress_empty_columns: bool

property suppress_empty_rows: bool

property titles: List[*ViewTitleSelection*]

class TM1py.**Process**(*name: str*, *has_security_access: bool | None = False*, *ui_data: str = 'CubeAction=1511\x0cDataAction=1503\x0cCubeLogChanges=0\x0c'*, *parameters: Iterable = None*, *variables: Iterable = None*, *variables_ui_data: Iterable = None*, *prolog_procedure: str = ''*, *metadata_procedure: str = ''*, *data_procedure: str = ''*, *epilog_procedure: str = ''*, *datasource_type: str = 'None'*, *datasource_ascii_decimal_separator: str = '.'*, *datasource_ascii_delimiter_char: str = ';'*, *datasource_ascii_delimiter_type: str = 'Character'*, *datasource_ascii_header_records: int = 1*, *datasource_ascii_quote_character: str = ''*, *datasource_ascii_thousand_separator: str = ','*, *datasource_data_source_name_for_client: str = ''*, *datasource_data_source_name_for_server: str = ''*, *datasource_password: str = ''*, *datasource_user_name: str = ''*, *datasource_query: str = ''*, *datasource_uses_unicode: bool = True*, *datasource_view: str = ''*, *datasource_subset: str = ''*)

Abstraction of a TM1 Process.

IMPORTANT. doesn't work with Processes that were generated through the Wizard

AUTO_GENERATED_STATEMENTS = '#**Begin: Generated Statements***\r\n#****End: Generated Statements****\r\n'**

BEGIN_GENERATED_STATEMENTS = '#**Begin: Generated Statements***'**

END_GENERATED_STATEMENTS = '#**End: Generated Statements****'**

MAX_STATEMENTS = 16380

MAX_STATEMENTS_POST_11_8_015 = 100000

static add_generated_string_to_code(*code: str*) → str

add_parameter(*name: str*, *prompt: str*, *value: str | int | float*, *parameter_type: str | None = None*)

Parameters

- **name** –
- **prompt** –
- **value** –
- **parameter_type** – introduced in TM1 11 REST API, therefor optional. if Not given type is derived from value

Returns

add_variable(*name: str*, *variable_type: str*)

add variable to the process

Parameters

- **name** –

  –

- **variable_type** – 'String' or 'Numeric'

Returns

property body: str

property data_procedure: str

property datasource_ascii_decimal_separator: str

property datasource_ascii_delimiter_char: str

property datasource_ascii_delimiter_type: str

property datasource_ascii_header_records: int

property datasource_ascii_quote_character: str

property datasource_ascii_thousand_separator: str

property datasource_data_source_name_for_client: str

property datasource_data_source_name_for_server: str

property datasource_password: str

property datasource_query: str

property datasource_subset: str

property datasource_type: str

property datasource_user_name: str

property datasource_uses_unicode: bool

property datasource_view: str

drop_parameter_types()

property epilog_procedure: str

classmethod from_dict(*process_as_dict: Dict*) → *Process*

Parameters
process_as_dict – Dictionary, process as dictionary

Returns
an instance of this class

classmethod from_json(*process_as_json: str*) → *Process*

Parameters
process_as_json – response of /Processes('x')?$expand=*

Returns
an instance of this class

property has_security_access: bool

static max_statements(*version: str*)

property metadata_procedure: str

property name: str

property parameters: List

property prolog_procedure: str

remove_parameter(*name: str*)

remove_variable(*name: str*)

property variables: List

class TM1py.Rules(*rules: str*)

Abstraction of Rules on a cube.

rules_analytics
A collection of rulestatements, where each statement is stored in uppercase without linebreaks. comments are not included.

KEYWORDS = ['SKIPCHECK', 'FEEDSTRINGS', 'UNDEFVALS', 'FEEDERS']

property feeder_statements: List[str]

property feedstrings: bool

property has_feeders: bool

init_analytics()

property rule_statements: List[str]

property rules_analytics: List[str]

property skipcheck: bool

property text: str

property undefvals: bool

class TM1py.Sandbox(*name: str*, *include_in_sandbox_dimension: bool = True*, *loaded: bool = False*, *active: bool = False*, *queued: bool = False*)

Abstraction of a TM1 Sandbox

property body: str

classmethod from_dict(*sandbox_as_dict: Dict*) → *Sandbox*

Alternative constructor

Parameters
sandbox_as_dict – user as dict

Returns
an instance of this class

classmethod from_json(*sandbox_as_json: str*) → *Sandbox*

Alternative constructor

Parameters
sandbox_as_json – user as JSON string

Returns
sandbox, an instance of this class

property include_in_sandbox_dimension: bool

property name: str

class TM1py.Server(*server_as_dict: Dict*)

Abstraction of the TM1 Server

Notes
contains the information you get from http://localhost:5895/Servers no methods so far

class TM1py.Subset(*subset_name: str*, *dimension_name: str*, *hierarchy_name: str = None*, *alias: str = None*, *expression: str = None*, *elements: Iterable[str] = None*)

Abstraction of the TM1 Subset (dynamic and static)

add_elements(*elements: Iterable[str]*)

add Elements to static subsets :Parameters:

elements : list of element names

property alias: str

property body: str

same logic here as in TM1 : when subset has expression its dynamic, otherwise static

property body_as_dict: Dict

same logic here as in TM1 : when subset has expression its dynamic, otherwise static

property dimension_name: str

property elements: List[str]

property expression: str

classmethod from_dict(*subset_as_dict: Dict*) → *Subset*

classmethod from_json(*subset_as_json: str*) → *Subset*

Alternative constructor :Parameters:

subset_as_json
[string, JSON] representation of Subset as specified in CSDL

Returns
Subset : an instance of this class

property hierarchy_name: str

property is_dynamic: bool

property is_static: bool

property name: str

property type: str

class TM1py.**User**(*name: str*, *groups: Iterable[str]*, *friendly_name: str | None = None*, *password: str | None = None*, *user_type: UserType | str = None*, *enabled: bool = None*)

Abstraction of a TM1 User

add_group(*group_name: str*)

property body: str

construct_body() → str

construct body (json) from the class attributes :return: String, TM1 JSON representation of a user

property enabled: bool

property friendly_name: str

classmethod from_dict(*user_as_dict: Dict*) → *User*

Alternative constructor

Parameters
user_as_dict – user as dict

Returns
user, an instance of this class

classmethod from_json(*user_as_json: str*)

Alternative constructor

Parameters
user_as_json – user as JSON string

Returns
user, an instance of this class

property groups: List[str]

property is_admin: bool

property is_data_admin: bool

property is_ops_admin: bool

property is_security_admin: bool

property name: str

property password: str

remove_group(*group_name: str*)

property user_type: UserType

class TM1py.**View**(*cube: str*, *name: str*)

Abstraction of TM1 View serves as a parentclass for TM1py.Objects.MDXView and TM1py.Objects.NativeView

abstract body() → str

property cube: str

property mdx

property name: str

class TM1py.**ViewAxisSelection**(*dimension_name: str*, *subset:* Subset | *AnonymousSubset*)

Describes what is selected in a dimension on an axis. Can be a Registered Subset or an Anonymous Subset

property body: str

property body_as_dict: Dict

property dimension_name: str

property hierarchy_name: str

property subset: *Subset* | AnonymousSubset

class TM1py.**ViewTitleSelection**(*dimension_name: str*, *subset: AnonymousSubset* | Subset, *selected: str*)

Describes what is selected in a dimension on the view title. Can be a Registered Subset or an Anonymous Subset

property body: str

property dimension_name: str

property hierarchy_name: str

property selected: str

property subset: *Subset* | AnonymousSubset

Exceptions

class TM1py.Exceptions.Exceptions.**TM1pyTimeout**(*method: str*, *url: str*, *timeout: float*)

class TM1py.Exceptions.Exceptions.**TM1pyVersionException**(*function: str*, *required_version*)

class TM1py.Exceptions.Exceptions.**TM1pyNotAdminException**(*function: str*)

class TM1py.Exceptions.Exceptions.**TM1pyRestException**(*response: str*, *status_code: int*, *reason: str*, *headers: Mapping*)

Exception for failing REST operations

property headers

property reason

property response

property status_code

class TM1py.Exceptions.Exceptions.**TM1pyException**(*message*)

The default exception for TM1py

class TM1py.Exceptions.Exceptions.**TM1pyWriteFailureException**(*statuses: List[str]*, *error_log_files: List[str]*)

class TM1py.Exceptions.Exceptions.**TM1pyWritePartialFailureException**(*statuses: List[str]*, *error_log_files: List[str]*, *attempts: int*)

PYTHON MODULE INDEX

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

T

U

V

W

Z