TM1py Documentation

Release 1.11.3

Marius Wirtz

May 23, 2023

Contents

1 Requirements **3**

2 Optional Requirements **5**

3 Install **7**

4 Usage **9**
4.1 API Documentation 9

Python Module Index **85**

Index **87**

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 1

Requirements

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

CHAPTER 2

Optional Requirements

- pandas

CHAPTER 3

Install

without pandas

```
pip install tm1py
```

with pandas

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

CHAPTER 4

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: TM1py.Services.RestService.RestService*)

Service to handle Object Updates for TM1 CellAnnotations

create(*annotation: TM1py.Objects.Annotation.Annotation*, ***kwargs*) → requests.models.Response

create an Annotation

Parameters **annotation** – instance of TM1py.Annotation

create_many(*annotations: Iterable[TM1py.Objects.Annotation.Annotation], **kwargs*) → requests.models.Response

create an Annotation

Parameters **annotations** – instances of TM1py.Annotation

delete(*annotation_id: str*, ***kwargs*) → requests.models.Response

delete Annotation

Parameters **annotation_id** – string, the id of the annotation

get(*annotation_id: str*, ***kwargs*) → TM1py.Objects.Annotation.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[TM1py.Objects.Annotation.Annotation]

get all annotations from given cube as a List.

Parameters **cube_name** –

update(*annotation: TM1py.Objects.Annotation.Annotation*, ***kwargs*) → requests.models.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/stable/TM1py/Servi*

Service to Read and Write TM1 Applications

create(*application: Union[TM1py.Objects.Application.Application, TM1py.Objects.Application.DocumentApplication], private: bool = False, **kwargs*) → requests.models.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*) → requests.models.Response

Create DocumentApplication in TM1 from local file

Parameters

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

Returns

delete(*path: str, application_type: Union[str, TM1py.Objects.Application.ApplicationTypes], application_name: str, private: bool = False, **kwargs*) → requests.models.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: Union[str, TM1py.Objects.Application.ApplicationTypes], name: str, private: bool = False, **kwargs*) → bool

Check if application exists

Parameters

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

Returns

get(*path: str, application_type: Union[str, TM1py.Objects.Application.ApplicationTypes], name: str, private: bool = False, **kwargs*) → TM1py.Objects.Application.Application

Retrieve Planning Analytics Application

Parameters

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

Returns

get_document(*path: str, name: str, private: bool = False, **kwargs*) → TM1py.Objects.Application.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: Union[str, TM1py.Objects.Application.ApplicationTypes], application_name: str, new_application_name: str, private: bool = False, **kwargs*)

update(*application: Union[TM1py.Objects.Application.Application, TM1py.Objects.Application.DocumentApplication], private: bool = False, **kwargs*) → requests.models.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*) → requests.models.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*) → requests.models.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: TM1py.Services.RestService.RestService*)

Service to handle Read and Write operations to TM1 cubes

activate_transactionlog(**args, **kwargs*) → requests.models.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: Union[Iterable[T_co], str], dimensions: Iterable[str] = None, sandbox_name: str = None, element_separator: str = ',', hierarchy_separator: str = '&&', hierarchy_element_separator: str = '::', **kwargs*) → Dict[KT, VT]

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*) → requests.models.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_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: Union[str, mdxpy.mdx.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, **kwargs*) → requests.models.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*) → requests.models.Response

Delete a cellset

Parameters

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

Returns

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

end_changeset(*change_set: str*) → requests.models.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, **kwargs*) → TM1py.Utils.Utils.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_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: Union[str, mdxpy.mdx.MdxBuilder], top: int = None, skip: int = None, skip_zeros: bool = True, skip_consolidated_cells: bool = False, skip_rule_derived_cells: bool = False, csv_dialect: Optional[csv.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, **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. :return: String

execute_mdx_dataframe(*mdx: Union[str, mdxpy.mdx.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, **kwargs*) → pandas.core.frame.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 :return: Pandas Dataframe

execute_mdx_dataframe_pivot(*mdx: str, dropna: bool = False, fill_value: bool = None, sandbox_name: str = None*) → pandas.core.frame.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*, ***kwargs*) → pandas.core.frame.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*) → TM1py.Utils.Utils.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[KT, VT]

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*) → TM1py.Utils.Utils.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*) → TM1py.Utils.Utils.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[KT, VT]

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[Union[float, str]]

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: TM1py.Objects.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, **kwargs*) → TM1py.Utils.Utils.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
- **use_compact_json** – bool

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: Optional[csv.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, **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. :return: 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, **kwargs*) → pandas.core.frame.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. :return: 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*) → pandas.core.frame.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, **kwargs*) → pandas.core.frame.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*) → TM1py.Utils.Utils.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[KT, VT]

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*) → TM1py.Utils.Utils.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*) → TM1py.Utils.Utils.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[Union[float, str]]

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*) → TM1py.Utils.Utils.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_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_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: Optional[csv.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, **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: bool :param include_headers: bool :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: Optional[csv.Dialect] = None, line_separator: str = '\r\n', value_separator: str = ', ', sandbox_name: str = None, include_attributes: 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 :return: Raw format from TM1.

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, **kwargs*) → pandas.core.frame.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*) → pandas.core.frame.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, **kwargs*) → pandas.core.frame.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_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[KT, VT]

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*) → requests.models.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*) → TM1py.Utils.Utils.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[Union[float, str]]

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: Union[str, Iterable[T_co]] = None*, *dimensions: List[str] = None*, *sandbox_name: str = None*, *element_separator: str = ', '*, *hierarchy_separator: str = '&&'*, *hierarchy_element_separator: str = '::'*, ***kwargs*) → Union[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_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*) → requests.models.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: Union[Iterable[T_co], str], dimensions: Iterable[str] = None, sandbox_name: str = None, depth: int = 1, element_separator: str = ',', hierarchy_separator: str = '&&', hierarchy_element_separator: str = '::', **kwargs*) → Dict[KT, VT]

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: Union[Iterable[T_co], str], dimensions: Iterable[str] = None, sandbox_name: str = None, element_separator: str = ',', hierarchy_separator: str = '&&', hierarchy_element_separator: str = '::', **kwargs*) → Dict[KT, VT]

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*) → requests.models.Response

undo a changeset. Similar to rolling back transactions.

Returns Change set ID

update_cellset(*cellset_id: str, values: Iterable[T_co], sandbox_name: str = None, changeset: str = None, **kwargs*) → requests.models.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[KT, VT], 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[KT, VT] = None, remove_blob: bool = True, **kwargs*) → Optional[str]

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 :return: changeset or None

write_async(*cube_name: str, cells: Dict[KT, VT], 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[KT, VT] = None, **kwargs*) → Optional[str]

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: pandas.core.frame.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[KT, VT] = None, sum_numeric_duplicates: bool = True, remove_blob: bool = True, **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 :return: changeset or None

write_dataframe_async(*cube_name: str, data: pandas.core.frame.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, **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 kwargs: :return: Success: bool, Messages: list, ChangeSet: None

write_through_cellset(*cube_name: str, cellset_as_dict: Dict[KT, VT], 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[KT, VT], increment: bool = False, sandbox_name: str = None, precision: int = None, skip_non_updateable: bool = False, measure_dimension_elements: Dict[KT, VT] = None, is_attribute_cube: bool = None, **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 kwargs: :return: Success: bool, Messages: list, ChangeSet: None

write_value(*value: Union[str, float], cube_name: str, element_tuple: Iterable[T_co], dimensions: Iterable[str] = None, sandbox_name: str = None, **kwargs*) → requests.models.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[KT, VT], 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[T_co], 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: TM1py.Services.RestService.RestService*)

Service to handle Object Updates for TM1 Chores

activate(*chore_name: str*, ***kwargs*) → requests.models.Response

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

create(*chore: TM1py.Objects.Chore.Chore*, ***kwargs*) → requests.models.Response

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

deactivate(*chore_name: str*, ***kwargs*) → requests.models.Response

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

delete(*chore_name: str*, ***kwargs*) → requests.models.Response

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

execute_chore(*chore_name: str*, ***kwargs*) → requests.models.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*) → TM1py.Objects.Chore.Chore

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

get_all(***kwargs*) → List[TM1py.Objects.Chore.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[TM1py.Objects.Chore.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[TM1py.Objects.Chore.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.datetime*, ***kwargs*) → requests.models.Response

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

update(*chore: TM1py.Objects.Chore.Chore*, ***kwargs*)

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

update_or_create(*chore: TM1py.Objects.Chore.Chore*, ***kwargs*) → requests.models.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: TM1py.Services.RestService.RestService*)

Service to handle Object Updates for TM1 Cubes

check_rules(*cube_name: str*, ***kwargs*) → requests.models.Response

Check rules syntax for existing cube on TM1 Server

Parameters **cube_name** – name of a cube

Returns response

create(*cube: TM1py.Objects.Cube.Cube*, ***kwargs*) → requests.models.Response

create new cube on TM1 Server

Parameters **cube** – instance of TM1py.Cube

Returns response

cube_save_data(*cube_name: str*, ***kwargs*) → requests.models.Response

Serializes a cube by saving data updates

Parameters **cube_name** –

Returns Response

delete(*cube_name: str*, ***kwargs*) → requests.models.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*) → TM1py.Objects.Cube.Cube

get cube from TM1 Server

Parameters **cube_name** –

Returns instance of TM1py.Cube

get_all(***kwargs*) → List[TM1py.Objects.Cube.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[TM1py.Objects.Cube.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[TM1py.Objects.Cube.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*) → requests.models.Response
Load the cube into memory on the server

Parameters **cube_name** –

Returns

lock(*cube_name: str*, ***kwargs*) → requests.models.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[TM1py.Objects.Cube.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*) → requests.models.Response
Unload the cube from memory

Parameters **cube_name** –

Returns

unlock(*cube_name: str*, ***kwargs*) → requests.models.Response
Unlocks the cube to allow modifications

Parameters **cube_name** –

Returns

update(*cube: TM1py.Objects.Cube.Cube*, ***kwargs*) → requests.models.Response
Update existing cube on TM1 Server

Parameters **cube** – instance of TM1py.Cube

Returns response

update_or_create(*cube: TM1py.Objects.Cube.Cube*, ***kwargs*) → requests.models.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: TM1py.Services.RestService.RestService*)
Service to handle Object Updates for TM1 Dimensions

create(*dimension: TM1py.Objects.Dimension.Dimension*, ***kwargs*) → requests.models.Response
Create a dimension

Parameters **dimension** – instance of TM1py.Dimension

Returns response

create_element_attributes_through_ti(*dimension: TM1py.Objects.Dimension.Dimension*, ***kwargs*)
:param dimension. Instance of TM1py.Objects.Dimension class :return:

delete(*dimension_name: str*, ***kwargs*) → requests.models.Response
Delete a dimension

Parameters **dimension_name** – Name of the dimension

Returns

execute_mdx(*dimension_name: str*, *mdx: str*, ***kwargs*) → List[T]
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*) → TM1py.Objects.Dimension.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: TM1py.Objects.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: TM1py.Objects.Dimension.Dimension*, ***kwargs*)

update if exists else create

Parameters **dimension** –

Returns

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

class TM1py.**ElementService**(*rest: TM1py.Services.RestService.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*) → requests.models.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[TM1py.Objects.ElementAttribute.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[TM1py.Objects.Element.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: TM1py.Objects.Element.Element*, ***kwargs*) → requests.models.Response

create_element_attribute(*dimension_name: str*, *hierarchy_name: str*, *element_attribute: TM1py.Objects.ElementAttribute.ElementAttribute*, ***kwargs*) → requests.models.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*) → requests.models.Response

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

like AttrDelete

Parameters

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

Returns

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: Optional[int] = None, member_properties: Optional[Iterable[str]] = ('Name', 'Weight'), parent_properties: Optional[Iterable[str]] = ('Name', 'UniqueName'), element_properties: Optional[Iterable[str]] = ('Type', 'Level'), **kwargs*) → List[T]

: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*) → TM1py.Objects.Element.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*) → TM1py.Utils.Utils.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*) → TM1py.Utils.Utils.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: Union[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[TM1py.Objects.Element.Element]

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

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[TM1py.Objects.ElementAttribute.ElementAttribute]

Get element attributes from hierarchy

Parameters

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

Returns

get_element_identifiers(*dimension_name: str, hierarchy_name: str, elements: Union[str, List[str]], **kwargs*) → TM1py.Utils.Utils.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*) → TM1py.Utils.Utils.CaseAndSpaceInsensitiveDict

get_element_types_from_all_hierarchies(*dimension_name: str*, *skip_consolidations: bool = False*, ***kwargs*) → TM1py.Utils.Utils.CaseAndSpaceInsensitiveDict

get_elements(*dimension_name: str*, *hierarchy_name: str*, ***kwargs*) → List[TM1py.Objects.Element.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: Union[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*, ***kwargs*) → pandas.core.frame.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

Returns pandas DataFrame

get_elements_filtered_by_attribute(*dimension_name: str, hierarchy_name: str, attribute_name: str, attribute_value: Union[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[TM1py.Objects.Element.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[TM1py.Objects.Element.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[TM1py.Objects.Element.Element]

hierarchy_exists(*dimension_name*, *hierarchy_name*)

remove_edge(*dimension_name: str*, *hierarchy_name: str*, *parent: str*, *component: str*, ***kwargs*) → requests.models.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: TM1py.Objects.Element.Element*, ***kwargs*) → requests.models.Response

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

Service to interact with GIT

COMMON_PARAMETERS = {'author': 'Author', 'branch': 'Branch', 'config': 'Config', 'e

git_execute_plan(*plan_id: str*, ***kwargs*) → requests.models.Response

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

git_get_plans(***kwargs*) → List[TM1py.Objects.GitPlan.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*) → TM1py.Objects.Git.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*) → requests.models.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*) → requests.models.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*) → TM1py.Objects.Git.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() → TM1py.Objects.GitProject.TM1Project

summary

tm1project_put(*tm1_project: TM1py.Objects.GitProject.TM1Project*) → TM1py.Objects.GitProject.TM1Project

class TM1py.**HierarchyService**(*rest: TM1py.Services.RestService.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*) → requests.models.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[TM1py.Objects.ElementAttribute.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[TM1py.Objects.Element.Element], **kwargs*)

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

Parameters

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

Returns

create(*hierarchy: TM1py.Objects.Hierarchy.Hierarchy, **kwargs*)

Create a hierarchy in an existing dimension

Parameters **hierarchy** –

Returns

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

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

Parameters

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

Returns

get(*dimension_name: str, hierarchy_name: str, **kwargs*) → TM1py.Objects.Hierarchy.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_default_member(*dimension_name: str*, *hierarchy_name: str = None*, ***kwargs*) → Optional[str]

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_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*) → requests.models.Response

remove_edges_under_consolidation(*dimension_name: str*, *hierarchy_name: str*, *consolidation_element: str*, ***kwargs*) → List[requests.models.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: TM1py.Objects.Hierarchy.Hierarchy*, *keep_existing_attributes=False*, ***kwargs*) → List[requests.models.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*) → requests.models.Response

Update the default member of a hierarchy. Currently implemented through TI, since TM1 API does not supports default member updates yet.

Parameters

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

Returns

update_element_attributes(*hierarchy: TM1py.Objects.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: TM1py.Objects.Hierarchy.Hierarchy*, ***kwargs*)

update if exists else create

Parameters **Hierarchy** –

Returns

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

Service to Query and Cancel Threads in TM1

cancel_all_running_threads(***kwargs*) → list

cancel_thread(*thread_id: int*, ***kwargs*) → requests.models.Response

Kill a running thread

Parameters **thread_id** –

Returns

close_all_sessions(***kwargs*) → list

close_session(*session_id*, ***kwargs*) → requests.models.Response

disconnect_all_users(***kwargs*) → list

disconnect_user(*user_name: str*, ***kwargs*) → requests.models.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[TM1py.Objects.User.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[T]

get_threads(***kwargs*) → List[T]
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*) → pandas.core.frame.DataFrame

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

get_member_properties(*dimension_name: str = None*, *hierarchy_name: str = None*, *member_selection: collections.abc.Iterable = None*, *skip_consolidations: bool = True*, *attributes: collections.abc.Iterable = None*, *skip_parents: bool = False*, *level_names=None*, *parent_attribute: str = None*, *skip_weights=True*, *use_blob=False*, ***kwargs*) → pandas.core.frame.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: TM1py.Services.RestService.RestService*)
Service to handle Object Updates for TI Processes

compile(*name: str*, ***kwargs*) → List[T]
Compile a Process. Return List of Syntax errors.

Parameters **name** –

Returns

compile_process(*process: TM1py.Objects.Process.Process*, ***kwargs*) → List[T]
Compile a Process. Return List of Syntax errors.

Parameters **process** –

Returns

create(*process: TM1py.Objects.Process.Process*, ***kwargs*) → requests.models.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: TM1py.Objects.ProcessDebugBreakpoint.ProcessDebugBreakpoint*, ***kwargs*) → requests.models.Response

debug_add_breakpoints(*debug_id: str*, *break_points: Iterable[TM1py.Objects.ProcessDebugBreakpoint.ProcessDebugBreakpoint] = None*, ***kwargs*) → requests.models.Response

debug_continue(*debug_id: str*, ***kwargs*) → Dict[KT, VT]
Resumes execution until next breakpoint

debug_get_breakpoints(*debug_id: str*, ***kwargs*) → List[TM1py.Objects.ProcessDebugBreakpoint.ProcessDebugBreak

debug_get_current_breakpoint(*debug_id: str*, ***kwargs*) → TM1py.Objects.ProcessDebugBreakpoint.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*) → requests.structures.CaseInsensitiveDict

debug_process(*process_name: str*, *timeout: float = None*, ***kwargs*) → Dict[KT, VT]
Start debug session for specified process; debug session id is returned in response

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

debug_step_in(*debug_id: str*, ***kwargs*) → Dict[KT, VT]
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[KT, VT]
Resumes execution and runs until current process has finished.

debug_step_over(*debug_id: str*, ***kwargs*) → Dict[KT, VT]
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: TM1py.Objects.ProcessDebugBreakpoint.ProcessDebugBreakpoi*
***kwargs*) → requests.models.Response

delete(*name: str*, ***kwargs*) → requests.models.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[KT, VT] = None, timeout: float = None, cancel_at_timeout: bool = False, **kwargs*) → requests.models.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: TM1py.Objects.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*) → requests.models.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, timeout: float = None, cancel_at_timeout: 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
- **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*) → TM1py.Objects.Process.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[TM1py.Objects.Process.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_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[T]
Get all ProcessErrorLog entries for a process

Parameters **process_name** – name of the process

Returns list - Collection of ProcessErrorLogs

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[T_co] = 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: TM1py.Objects.Process.Process, **kwargs*) → requests.models.Response

Update an existing Process on TM1 Server

Parameters **process** – Instance of TM1py.Process class

Returns Response

update_or_create(*process: TM1py.Objects.Process.Process, **kwargs*) → requests.models.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

DELETE(*url: str, data: Union[str, bytes], headers: Dict[KT, VT] = None, timeout: float = None, **kwargs*)

Delete request against TM1 instance :param url: String, for instance :

/api/v1/Dimensions('plan_business_unit') :param data: the payload :param headers: custom headers :param timeout: Number of seconds that the client will wait to receive the first byte. :return: response object

GET(*url: str, data: Union[str, bytes] = '', headers: Dict[KT, VT] = None, timeout: float = None, **kwargs*)
Perform a GET request against TM1 instance :param url: :param data: the payload :param headers: custom headers :param timeout: Number of seconds that the client will wait to receive the first byte. :return: response object

HEADERS = {'Accept': 'application/json;odata.metadata=none,text/plain', 'Connection':

PATCH(*url: str, data: Union[str, bytes], headers: Dict[KT, VT] = None, timeout: float = None, **kwargs*)
PATCH request against the TM1 instance :param url: String, for instance : /api/v1/Dimensions('plan_business_unit') :param data: the payload :param headers: custom headers :param timeout: Number of seconds that the client will wait to receive the first byte. :return: response object

POST(*url: str, data: Union[str, bytes], headers: Dict[KT, VT] = None, timeout: float = None, **kwargs*)
POST request against the TM1 instance :param url: :param data: the payload :param headers: custom headers :param timeout: Number of seconds that the client will wait to receive the first byte. :return: response object

PUT(*url: str, data: Union[str, bytes], headers: Dict[KT, VT] = None, timeout: float = None, **kwargs*)
PUT request against the TM1 instance :param url: String, for instance : /api/v1/Dimensions('plan_business_unit') :param data: the payload :param headers: custom headers :param timeout: Number of seconds that the client will wait to receive the first byte. :return: response object

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_raw_bytes(*data: bytes*) → requests.models.Response

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

cancel_running_operation()

connect()

static disable_http_warnings()

get_http_header(*key: str*) → str

get_monitoring_service()

is_admin

is_connected() → bool
Check if Connection to TM1 Server is established. :Returns:

Boolean

logout(*timeout: float = None, **kwargs*)
End TM1 Session and HTTP session

remove_http_header(*key: str*)

retrieve_async_response(*async_id: str*, ***kwargs*) → requests.models.Response

sandboxing_disabled

session_id

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*) → http.client.HTTPResponse
Build urllib3.HTTPResponse based on raw bytes string

static verify_response(*response: requests.models.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.

version

static wait_time_generator(*timeout: int*)

class TM1py.**SandboxService**(*rest: TM1py.Services.RestService.RestService*)
Service to handle sandboxes in TM1

create(*sandbox: TM1py.Objects.Sandbox.Sandbox*, ***kwargs*) → requests.models.Response
create a new sandbox in TM1 Server

Parameters **sandbox** – Sandbox

Returns response

delete(*sandbox_name: str*, ***kwargs*) → requests.models.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*) → TM1py.Objects.Sandbox.Sandbox
get a sandbox from TM1 Server

Parameters **sandbox_name** – str

Returns instance of TM1py.Sandbox

get_all(***kwargs*) → List[TM1py.Objects.Sandbox.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*) → requests.models.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*) → requests.models.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*) → requests.models.Response
publish existing sandbox to base

Parameters **sandbox_name** – str

Returns response

reset(*sandbox_name: str*, ***kwargs*) → requests.models.Response
reset all changes in specified sandbox

Parameters **sandbox_name** – str

Returns response

unload(*sandbox_name: str*, ***kwargs*) → requests.models.Response
unload sandbox from memory

Parameters **sandbox_name** – str

Returns response

update(*sandbox: TM1py.Objects.Sandbox.Sandbox*, ***kwargs*) → requests.models.Response
update a sandbox in TM1

Parameters **sandbox** –

Returns response

class TM1py.**SecurityService**(*rest: TM1py.Services.RestService.RestService*)
Service to handle Security stuff

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

Parameters

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

Returns response

create_group(*group_name: str*, ***kwargs*) → requests.models.Response
Create a Security group in the TM1 Server

Parameters **group_name** –

Returns

create_user(*user: TM1py.Objects.User.User*, ***kwargs*) → requests.models.Response
Create a user on TM1 Server

Parameters **user** – instance of TM1py.User

Returns response

delete_group(*group_name: str*, ***kwargs*) → requests.models.Response
Delete a group in the TM1 Server

Parameters **group_name** –

Returns

delete_user(*user_name: str*, ***kwargs*) → requests.models.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*) → TM1py.Objects.User.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*) → TM1py.Objects.User.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*) → requests.models.Response
Remove user from group in TM1 Server

Parameters

- **group_name** –
- **user_name** –

Returns response

security_refresh(***kwargs*) → requests.models.Response

update_user(*user: TM1py.Objects.User.User*, ***kwargs*) → requests.models.Response
Update user on TM1 Server

Parameters **user** – instance of TM1py.User

Returns response

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

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

class TM1py.**ServerService**(*rest: TM1py.Services.RestService.RestService*)
Service to query common information from the TM1 Server

activate_audit_log()

deactivate_audit_log()

delete_persistent_feeders(***kwargs*) → requests.models.Response

execute_audit_log_delta_request(***kwargs*) → Dict[KT, VT]

execute_message_log_delta_request(***kwargs*) → Dict[KT, VT]

execute_transaction_log_delta_request(***kwargs*) → Dict[KT, VT]

get_active_configuration(***kwargs*) → Dict[KT, VT]
Read effective(!) TM1 config settings as dictionary from TM1 Server

Returns config as dictionary

get_admin_host(***kwargs*) → str

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

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[KT, VT]

get_data_directory(***kwargs*) → str

get_last_process_message_from_messagelog(*process_name: str*, ***kwargs*) → Optional[str]

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.datetime = None*, *until: datetime.datetime = None*, *top: int = None*, *logger: str = None*, *level: str = None*, *msg_contains: collections.abc.Iterable = None*, *msg_contains_operator: str = 'and'*, ***kwargs*) → Dict[KT, VT]

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[KT, VT]

Read TM1 config settings as dictionary from TM1 Server

Returns config as dictionary

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

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*) → requests.models.Response

start_performance_monitor()

stop_performance_monitor()

update_static_configuration(*configuration: Dict[KT, VT]*) → requests.models.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: TM1py.Services.RestService.RestService*)

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

create(*subset: TM1py.Objects.Subset.Subset, private: bool = False, **kwargs*) → requests.models.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*) → requests.models.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*) → requests.models.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*) → TM1py.Objects.Subset.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*) → requests.models.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: TM1py.Objects.Subset.Subset*, *private: bool = False*, ***kwargs*) → requests.models.Response

update a subset on the TM1 Server

Parameters

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

Returns response

update_or_create(*subset: TM1py.Objects.Subset.Subset*, *private: bool = False*, ***kwargs*) → requests.models.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.

connection

logout(***kwargs*)

re_authenticate()

re_connect()

classmethod restore_from_file(*file_name*)

save_to_file(*file_name*)

version

whoami

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

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

create(*view: Union[TM1py.Objects.MDXView.MDXView, TM1py.Objects.NativeView.NativeView], private: bool = False, **kwargs*) → requests.models.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*) → requests.models.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*) → TM1py.Objects.View.View

get_all(*cube_name: str, include_elements: bool = True, **kwargs*) → Tuple[List[TM1py.Objects.View.View], List[TM1py.Objects.View.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*) → TM1py.Objects.MDXView.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*) → TM1py.Objects.NativeView.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[TM1py.Objects.View.View], List[TM1py.Objects.View.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: Union[TM1py.Objects.MDXView.MDXView, TM1py.Objects.NativeView.NativeView], private: bool = False, **kwargs*) → requests.models.Response

Update an existing view

Parameters

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

Returns response

update_or_create(*view: Union[TM1py.Objects.MDXView.MDXView, TM1py.Objects.NativeView.NativeView], private: bool = False, **kwargs*) → requests.models.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

body

body_as_dict

comment_value

construct_body_for_post(*cube_dimensions*) → Dict[KT, VT]

created

dimensional_context

classmethod from_json(*annotation_as_json: str*) → TM1py.Objects.Annotation.Annotation

Alternative constructor

Parameters **annotation_as_json** – String, JSON

Returns instance of TM1py.Process

id

last_updated

last_updated_by

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

object_name

text

class TM1py.**Application**(*path: str, name: str, application_type: Union[TM1py.Objects.Application.ApplicationTypes, str]*)

application_id

body

body_as_dict

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

Abstraction of TM1 Chore

MULTIPLE_COMMIT = 'MultipleCommit'

SINGLE_COMMIT = 'SingleCommit'

activate()

active

add_task(*task: TM1py.Objects.ChoreTask.ChoreTask*)

body

body_as_dict

construct_body() → str

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

deactivate()

dst_sensitivity

execution_mode

frequency

classmethod from_dict(*chore_as_dict: Dict[KT, VT]*) → TM1py.Objects.Chore.Chore

Alternative constructor

Parameters **chore_as_dict** – Chore as dict

Returns Chore, an instance of this class

classmethod from_json(*chore_as_json: str*) → TM1py.Objects.Chore.Chore

Alternative constructor

Parameters **chore_as_json** – string, JSON. Response of /api/v1/Chores('x')/Tasks?$expand=*

Returns Chore, an instance of this class

name

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

start_time

tasks

class TM1py.**ChoreFrequency**(*days: Union[str, int], hours: Union[str, int], minutes: Union[str, int], seconds: Union[str, int]*)

Utility class to handle time representation fore Chore Frequency

days

frequency_string

classmethod from_string(*frequency_string: str*) → TM1py.Objects.ChoreFrequency.ChoreFrequency

hours

minutes

seconds

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*)

datetime

classmethod from_string(*start_time_string: str*) → TM1py.Objects.ChoreStartTime.ChoreStartTime

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

start_time_string

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 coniststof - 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

body

body_as_dict

classmethod from_dict(*chore_task_as_dict: Dict[KT, VT]*)

parameters

process_name

step

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

Abstraction of a TM1 Cube

body

dimensions

feedstrings

classmethod from_dict(*cube_as_dict: Dict[KT, VT]*) → TM1py.Objects.Cube.Cube

Alternative constructor

Parameters **cube_as_dict** – user as dict

Returns user, an instance of this class

classmethod from_json(*cube_as_json: str*) → TM1py.Objects.Cube.Cube

Alternative constructor

Parameters **cube_as_json** – user as JSON string

Returns cube, an instance of this class

has_rules

name

rules

skipcheck

undefvals

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

Abstraction of TM1 Dimension

A Dimension is a container for hierarchies.

add_hierarchy(*hierarchy: TM1py.Objects.Hierarchy.Hierarchy*)

body

body_as_dict

contains_hierarchy(*hierarchy_name: str*) → bool

default_hierarchy

classmethod from_dict(*dimension_as_dict: Dict[KT, VT]*) → TM1py.Objects.Dimension.Dimension

classmethod from_json(*dimension_as_json: str*) → TM1py.Objects.Dimension.Dimension

get_hierarchy(*hierarchy_name: str*) → TM1py.Objects.Hierarchy.Hierarchy

hierarchies

hierarchy_names

name

remove_hierarchy(*hierarchy_name: str*)

unique_name

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

Abstraction of TM1 Element

ELEMENT_ATTRIBUTES_PREFIX = '}ElementAttributes_'

class Types

An enumeration.

CONSOLIDATED = 3

NUMERIC = 1

STRING = 2

body

body_as_dict

element_attributes

element_type

static from_dict(*element_as_dict: Dict[KT, VT]*) → TM1py.Objects.Element.Element

index

name

unique_name

class TM1py.**ElementAttribute**(*name: str, attribute_type: Union[TM1py.Objects.ElementAttribute.ElementAttribute.Types, str]*)

Abstraction of TM1 Element Attributes

class Types

An enumeration.

ALIAS = 3

NUMERIC = 1

STRING = 2

attribute_type

body

body_as_dict

classmethod from_dict(*element_attribute_as_dict: Dict[KT, VT]*) → TM1py.Objects.ElementAttribute.ElementAttribute

classmethod from_json(*element_attribute_as_json: str*) → TM1py.Objects.ElementAttribute.ElementAttribute

name

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

Abstraction of Git object

config

deployed_commit

deployment

force

classmethod from_dict(*json_response: Dict[KT, VT]*) → TM1py.Objects.Git.Git

remote

url

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

Abstraction of Git Commit

author

commit_id

summary

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

Base GitPlan abstraction

branch

force

plan_id

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

Abstraction of GitRemote

branches

connected

tags

class TM1py.**Hierarchy**(*name: str, dimension_name: str, elements: Optional[Iterable[Element]] = None, element_attributes: Optional[Iterable[ElementAttribute]] = None, edges: Optional[Dict] = None, subsets: Optional[Iterable[str]] = None, structure: Optional[int] = None, default_member: Optional[str] = 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: Union[str, TM1py.Objects.Element.Element.Types]*)

add_element_attribute(*name: str, attribute_type: str*)

balanced

body

body_as_dict

contains_element(*element_name: str*) → bool

default_member

dimension_name

edges

element_attributes

elements

classmethod from_dict(*hierarchy_as_dict: Dict[KT, VT], dimension_name: str = None*) → TM1py.Objects.Hierarchy.Hierarchy

get_ancestors(*element_name: str, recursive: bool = False*) → Set[TM1py.Objects.Element.Element]

get_descendant_edges(*element_name: str, recursive: bool = False*) → Dict[KT, VT]

get_descendants(*element_name: str, recursive: bool = False, leaves_only=False*) → Set[TM1py.Objects.Element.Element]

get_element(*element_name: str*) → TM1py.Objects.Element.Element

name

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*)

subsets

update_edge(*parent: str*, *component: str*, *weight: float*)

update_element(*element_name: str, element_type: Union[str, TM1py.Objects.Element.Element.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!

MDX

body

construct_body() → str

classmethod from_dict(*view_as_dict: Dict[KT, VT], cube_name: str = None*) → TM1py.Objects.MDXView.MDXView

classmethod from_json(*view_as_json: str, cube_name: Optional[str] = None*) → TM1py.Objects.MDXView.MDXView

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: Optional[bool] = False, suppress_empty_rows: Optional[bool] = False, format_string: Optional[str] = '0.########', titles: Optional[Iterable[TM1py.Objects.Axis.ViewTitleSelection]] = None, columns: Optional[Iterable[TM1py.Objects.Axis.ViewAxisSelection]] = None, rows: Optional[Iterable[TM1py.Objects.Axis.ViewAxisSelection]] = None*)

Abstraction of TM1 NativeView (classic cube view)

Notes Complete, functional and tested

MDX

add_column(*dimension_name: str, subset: Union[TM1py.Objects.Subset.Subset, TM1py.Objects.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: TM1py.Objects.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: Union[TM1py.Objects.Subset.Subset, TM1py.Objects.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

as_MDX

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

body

columns

format_string

classmethod from_dict(*view_as_dict: Dict[KT, VT]*, *cube_name: str = None*) → TM1py.Objects.NativeView.NativeView

classmethod from_json(*view_as_json: str*, *cube_name: Optional[str] = None*) → TM1py.Objects.NativeView.NativeView

Alternative constructor :Parameters:

view_as_json : string, JSON

Returns *View* : an instance of this class

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

rows

substitute_title(*dimension: str*, *element: str*)

suppress_empty_cells

suppress_empty_columns

suppress_empty_rows

titles

class TM1py.**Process**(*name: str*, *has_security_access: Optional[bool] = False*, *ui_data: str = 'CubeAction=1511x0cDataAction=1503x0cCubeLogChanges=0x0c'*, *parameters: Iterable[T_co] = None*, *variables: Iterable[T_co] = None*, *variables_ui_data: Iterable[T_co] = 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: Genera**

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: Union[str, int, float], parameter_type: Optional[str] = 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

body

data_procedure

datasource_ascii_decimal_separator

datasource_ascii_delimiter_char

datasource_ascii_delimiter_type

datasource_ascii_header_records

datasource_ascii_quote_character

datasource_ascii_thousand_separator

datasource_data_source_name_for_client

datasource_data_source_name_for_server

datasource_password

datasource_query

datasource_subset

datasource_type

datasource_user_name

datasource_uses_unicode

datasource_view

drop_parameter_types()

epilog_procedure

classmethod from_dict(*process_as_dict: Dict[KT, VT]*) → TM1py.Objects.Process.Process

Parameters **process_as_dict** – Dictionary, process as dictionary

Returns an instance of this class

classmethod from_json(*process_as_json: str*) → TM1py.Objects.Process.Process

Parameters **process_as_json** – response of /api/v1/Processes('x')?$expand=*

Returns an instance of this class

has_security_access

static max_statements(*version: str*)

metadata_procedure

name

parameters

prolog_procedure

remove_parameter(*name: str*)

remove_variable(*name: str*)

variables

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']

feeder_statements

feedstrings

has_feeders

init_analytics()

rule_statements

rules_analytics

skipcheck

text

undefvals

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

body

classmethod from_dict(*sandbox_as_dict: Dict[KT, VT]*) → TM1py.Objects.Sandbox.Sandbox

Alternative constructor

Parameters **sandbox_as_dict** – user as dict

Returns an instance of this class

classmethod from_json(*sandbox_as_json: str*) → TM1py.Objects.Sandbox.Sandbox

Alternative constructor

Parameters **sandbox_as_json** – user as JSON string

Returns sandbox, an instance of this class

include_in_sandbox_dimension

name

class TM1py.**Server**(*server_as_dict: Dict[KT, VT]*)

Abstraction of the TM1 Server

Notes contains the information you get from http://localhost:5895/api/v1/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

alias

body

same logic here as in TM1 : when subset has expression its dynamic, otherwise static

body_as_dict

same logic here as in TM1 : when subset has expression its dynamic, otherwise static

dimension_name

elements

expression

classmethod from_dict(*subset_as_dict: Dict[KT, VT]*) → TM1py.Objects.Subset.Subset

classmethod from_json(*subset_as_json: str*) → TM1py.Objects.Subset.Subset

Alternative constructor :Parameters:

subset_as_json [string, JSON] representation of Subset as specified in CSDL

Returns *Subset* : an instance of this class

hierarchy_name

is_dynamic

is_static

name

type

class TM1py.**User**(*name: str, groups: Iterable[str], friendly_name: Optional[str] = None, password: Optional[str] = None, user_type: Union[TM1py.Objects.User.UserType, str] = None, enabled: bool = None*)

Abstraction of a TM1 User

add_group(*group_name: str*)

body

construct_body() → str

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

enabled

friendly_name

classmethod from_dict(*user_as_dict: Dict[KT, VT]*) → TM1py.Objects.User.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

groups

is_admin

name

password

remove_group(*group_name: str*)

user_type

class TM1py.**View**(*cube: str*, *name: str*)

Abstraction of TM1 View serves as a parentclass for TM1py.Objects.MDXView and TM1py.Objects.NativeView

body() → str

cube

mdx

name

class TM1py.**ViewAxisSelection**(*dimension_name: str, subset: Union[TM1py.Objects.Subset.Subset, TM1py.Objects.Subset.AnonymousSubset]*)

Describes what is selected in a dimension on an axis. Can be a Registered Subset or an Anonymous Subset

body

body_as_dict

dimension_name

hierarchy_name

subset

class TM1py.**ViewTitleSelection**(*dimension_name: str, subset: Union[TM1py.Objects.Subset.AnonymousSubset, TM1py.Objects.Subset.Subset], selected: str*)

Describes what is selected in a dimension on the view title. Can be a Registered Subset or an Anonymous Subset

body

dimension_name

hierarchy_name

selected

subset

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[KT, VT_co]*)

Exception for failing REST operations

headers

reason

response

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

S

schedule, 9

Index

A

activate() (*TM1py.Chore method*), 71
activate() (*TM1py.ChoreService method*), 38
activate_audit_log() (*TM1py.ServerService method*), 63
activate_transactionlog() (*TM1py.CellService method*), 12
active (*TM1py.Chore attribute*), 71
add() (*TM1py.ChoreStartTime method*), 72
add_column() (*TM1py.NativeView method*), 76
add_compact_json_header() (*TM1py.RestService method*), 59
add_component() (*TM1py.Hierarchy method*), 75
add_edge() (*TM1py.Hierarchy method*), 75
add_edges() (*TM1py.ElementService method*), 43
add_edges() (*TM1py.HierarchyService method*), 50
add_element() (*TM1py.Hierarchy method*), 75
add_element_attribute() (*TM1py.Hierarchy method*), 75
add_element_attributes() (*TM1py.ElementService method*), 43
add_element_attributes() (*TM1py.HierarchyService method*), 51
add_elements() (*TM1py.ElementService method*), 43
add_elements() (*TM1py.HierarchyService method*), 51
add_elements() (*TM1py.Subset method*), 81
add_generated_string_to_code() (*TM1py.Process static method*), 78
add_group() (*TM1py.User method*), 81
add_hierarchy() (*TM1py.Dimension method*), 73
add_http_header() (*TM1py.RestService method*), 59
add_parameter() (*TM1py.Process method*), 78
add_row() (*TM1py.NativeView method*), 77
add_task() (*TM1py.Chore method*), 71
add_title() (*TM1py.NativeView method*), 77
add_user_to_groups() (*TM1py.SecurityService method*), 61
add_variable() (*TM1py.Process method*), 79
ALIAS (*TM1py.ElementAttribute.Types attribute*), 74
alias (*TM1py.Subset attribute*), 81
Annotation (*class in TM1py*), 70
AnnotationService (*class in TM1py*), 10
Application (*class in TM1py*), 70
application_id (*TM1py.Application attribute*), 70
ApplicationService (*class in TM1py*), 10
as_MDX (*TM1py.NativeView attribute*), 77
attribute_cube_exists() (*TM1py.ElementService method*), 44
attribute_type (*TM1py.ElementAttribute attribute*), 74
author (*TM1py.GitCommit attribute*), 74
AUTO_GENERATED_STATEMENTS (*TM1py.Process attribute*), 78

B

b64_decode_password() (*TM1py.RestService static method*), 59
balanced (*TM1py.Hierarchy attribute*), 75
begin_changeset() (*TM1py.CellService method*), 12
BEGIN_GENERATED_STATEMENTS (*TM1py.Process attribute*), 78
body (*TM1py.Annotation attribute*), 70
body (*TM1py.Application attribute*), 70
body (*TM1py.Chore attribute*), 71
body (*TM1py.ChoreTask attribute*), 72
body (*TM1py.Cube attribute*), 72
body (*TM1py.Dimension attribute*), 73
body (*TM1py.Element attribute*), 73
body (*TM1py.ElementAttribute attribute*), 74
body (*TM1py.Hierarchy attribute*), 75
body (*TM1py.MDXView attribute*), 76
body (*TM1py.NativeView attribute*), 77
body (*TM1py.Process attribute*), 79
body (*TM1py.Sandbox attribute*), 80
body (*TM1py.Subset attribute*), 81

body (*TM1py.User attribute*), 81
body (*TM1py.ViewAxisSelection attribute*), 82
body (*TM1py.ViewTitleSelection attribute*), 82
body() (*TM1py.View method*), 82
body_as_dict (*TM1py.Annotation attribute*), 70
body_as_dict (*TM1py.Application attribute*), 70
body_as_dict (*TM1py.Chore attribute*), 71
body_as_dict (*TM1py.ChoreTask attribute*), 72
body_as_dict (*TM1py.Dimension attribute*), 73
body_as_dict (*TM1py.Element attribute*), 73
body_as_dict (*TM1py.ElementAttribute attribute*), 74
body_as_dict (*TM1py.Hierarchy attribute*), 75
body_as_dict (*TM1py.Subset attribute*), 81
body_as_dict (*TM1py.ViewAxisSelection attribute*), 82
branch (*TM1py.GitPlan attribute*), 74
branches (*TM1py.GitRemote attribute*), 74
build_response_from_raw_bytes() (*TM1py.RestService static method*), 59

C

cancel_all_running_threads() (*TM1py.MonitoringService method*), 53
cancel_async_operation() (*TM1py.RestService method*), 59
cancel_running_operation() (*TM1py.RestService method*), 59
cancel_thread() (*TM1py.MonitoringService method*), 53
CellService (*class in TM1py*), 12
check_cell_feeders() (*TM1py.CellService method*), 12
check_rules() (*TM1py.CubeService method*), 39
Chore (*class in TM1py*), 70
ChoreFrequency (*class in TM1py*), 71
ChoreService (*class in TM1py*), 38
ChoreStartTime (*class in TM1py*), 71
ChoreTask (*class in TM1py*), 72
clear() (*TM1py.CellService method*), 13
clear_spread() (*TM1py.CellService method*), 13
clear_with_mdx() (*TM1py.CellService method*), 13
close_all_sessions() (*TM1py.MonitoringService method*), 53
close_session() (*TM1py.MonitoringService method*), 53
columns (*TM1py.NativeView attribute*), 77
comment_value (*TM1py.Annotation attribute*), 70
commit_id (*TM1py.GitCommit attribute*), 74
COMMON_PARAMETERS (*TM1py.GitService attribute*), 49
compile() (*TM1py.ProcessService method*), 54
compile_process() (*TM1py.ProcessService method*), 54
config (*TM1py.Git attribute*), 74
connect() (*TM1py.RestService method*), 59
connected (*TM1py.GitRemote attribute*), 74
connection (*TM1py.TM1Service attribute*), 67
CONSOLIDATED (*TM1py.Element.Types attribute*), 73
construct_body() (*TM1py.Chore method*), 71
construct_body() (*TM1py.MDXView method*), 76
construct_body() (*TM1py.User method*), 81
construct_body_for_post() (*TM1py.Annotation method*), 70
contains_element() (*TM1py.Hierarchy method*), 75
contains_hierarchy() (*TM1py.Dimension method*), 73
create() (*TM1py.AnnotationService method*), 10
create() (*TM1py.ApplicationService method*), 10
create() (*TM1py.ChoreService method*), 38
create() (*TM1py.CubeService method*), 39
create() (*TM1py.DimensionService method*), 42
create() (*TM1py.ElementService method*), 44
create() (*TM1py.HierarchyService method*), 51
create() (*TM1py.ProcessService method*), 55
create() (*TM1py.SandboxService method*), 60
create() (*TM1py.SubsetService method*), 65
create() (*TM1py.ViewService method*), 68
create_cellset() (*TM1py.CellService method*), 13
create_cellset_from_view() (*TM1py.CellService method*), 14
create_document_from_file() (*TM1py.ApplicationService method*), 10
create_element_attribute() (*TM1py.ElementService method*), 44
create_element_attributes_through_ti() (*TM1py.DimensionService method*), 42
create_group() (*TM1py.SecurityService method*), 61
create_many() (*TM1py.AnnotationService method*), 10
create_user() (*TM1py.SecurityService method*), 62
created (*TM1py.Annotation attribute*), 70
Cube (*class in TM1py*), 72
cube (*TM1py.View attribute*), 82
cube_save_data() (*TM1py.CubeService method*), 39
CubeService (*class in TM1py*), 39

D

data_procedure (*TM1py.Process attribute*), 79
datasource_ascii_decimal_separator (*TM1py.Process attribute*), 79
datasource_ascii_delimiter_char (*TM1py.Process attribute*), 79
datasource_ascii_delimiter_type (*TM1py.Process attribute*), 79

datasource_ascii_header_records (*TM1py.Process attribute*), 79
datasource_ascii_quote_character (*TM1py.Process attribute*), 79
datasource_ascii_thousand_separator (*TM1py.Process attribute*), 79
datasource_data_source_name_for_client (*TM1py.Process attribute*), 79
datasource_data_source_name_for_server (*TM1py.Process attribute*), 79
datasource_password (*TM1py.Process attribute*), 79
datasource_query (*TM1py.Process attribute*), 79
datasource_subset (*TM1py.Process attribute*), 79
datasource_type (*TM1py.Process attribute*), 79
datasource_user_name (*TM1py.Process attribute*), 79
datasource_uses_unicode (*TM1py.Process attribute*), 79
datasource_view (*TM1py.Process attribute*), 79
datetime (*TM1py.ChoreStartTime attribute*), 72
days (*TM1py.ChoreFrequency attribute*), 71
deactivate() (*TM1py.Chore method*), 71
deactivate() (*TM1py.ChoreService method*), 38
deactivate_audit_log() (*TM1py.ServerService method*), 63
deactivate_transactionlog() (*TM1py.CellService method*), 14
debug_add_breakpoint() (*TM1py.ProcessService method*), 55
debug_add_breakpoints() (*TM1py.ProcessService method*), 55
debug_continue() (*TM1py.ProcessService method*), 55
debug_get_breakpoints() (*TM1py.ProcessService method*), 55
debug_get_current_breakpoint() (*TM1py.ProcessService method*), 55
debug_get_process_line_number() (*TM1py.ProcessService method*), 55
debug_get_process_procedure() (*TM1py.ProcessService method*), 55
debug_get_record_number() (*TM1py.ProcessService method*), 55
debug_get_single_variable_value() (*TM1py.ProcessService method*), 55
debug_get_variable_values() (*TM1py.ProcessService method*), 55
debug_process() (*TM1py.ProcessService method*), 55
debug_remove_breakpoint() (*TM1py.ProcessService method*), 55
debug_step_in() (*TM1py.ProcessService method*), 55
debug_step_out() (*TM1py.ProcessService method*), 55
debug_step_over() (*TM1py.ProcessService method*), 55
debug_update_breakpoint() (*TM1py.ProcessService method*), 55
default_hierarchy (*TM1py.Dimension attribute*), 73
default_member (*TM1py.Hierarchy attribute*), 75
delete() (*TM1py.AnnotationService method*), 10
delete() (*TM1py.ApplicationService method*), 11
delete() (*TM1py.ChoreService method*), 38
delete() (*TM1py.CubeService method*), 39
delete() (*TM1py.DimensionService method*), 42
delete() (*TM1py.ElementService method*), 44
delete() (*TM1py.HierarchyService method*), 51
delete() (*TM1py.ProcessService method*), 55
DELETE() (*TM1py.RestService method*), 58
delete() (*TM1py.SandboxService method*), 60
delete() (*TM1py.SubsetService method*), 66
delete() (*TM1py.ViewService method*), 68
delete_cellset() (*TM1py.CellService method*), 14
delete_element_attribute() (*TM1py.ElementService method*), 44
delete_elements_from_static_subset() (*TM1py.SubsetService method*), 66
delete_group() (*TM1py.SecurityService method*), 62
delete_persistent_feeders() (*TM1py.ServerService method*), 63
delete_user() (*TM1py.SecurityService method*), 62
deployed_commit (*TM1py.Git attribute*), 74
deployment (*TM1py.Git attribute*), 74
determine_actual_group_name() (*TM1py.SecurityService method*), 62
determine_actual_user_name() (*TM1py.SecurityService method*), 62
Dimension (*class in TM1py*), 73
dimension_name (*TM1py.Hierarchy attribute*), 75
dimension_name (*TM1py.Subset attribute*), 81
dimension_name (*TM1py.ViewAxisSelection attribute*), 82
dimension_name (*TM1py.ViewTitleSelection attribute*), 82
dimensional_context (*TM1py.Annotation attribute*), 70
dimensions (*TM1py.Cube attribute*), 72
DimensionService (*class in TM1py*), 42
disable_http_warnings() (*TM1py.RestService static method*), 59
disconnect_all_users() (*TM1py.MonitoringService method*), 53
disconnect_user() (*TM1py.MonitoringService method*), 53

drop_non_updateable_cells() (*TM1py.CellService method*), 14
drop_parameter_types() (*TM1py.Process method*), 79
dst_sensitivity (*TM1py.Chore attribute*), 71

E

edges (*TM1py.Hierarchy attribute*), 75
EDGES_WORKAROUND_VERSIONS (*TM1py.HierarchyService attribute*), 50
Element (*class in TM1py*), 73
Element.Types (*class in TM1py*), 73
element_attributes (*TM1py.Element attribute*), 73
element_attributes (*TM1py.Hierarchy attribute*), 75
ELEMENT_ATTRIBUTES_PREFIX (*TM1py.Element attribute*), 73
element_is_ancestor() (*TM1py.ElementService method*), 44
element_is_parent() (*TM1py.ElementService method*), 44
element_type (*TM1py.Element attribute*), 73
ElementAttribute (*class in TM1py*), 73
ElementAttribute.Types (*class in TM1py*), 74
elements (*TM1py.Hierarchy attribute*), 75
elements (*TM1py.Subset attribute*), 81
ElementService (*class in TM1py*), 43
enabled (*TM1py.User attribute*), 81
end_changeset() (*TM1py.CellService method*), 14
END_GENERATED_STATEMENTS (*TM1py.Process attribute*), 78
epilog_procedure (*TM1py.Process attribute*), 79
evaluate_boolean_ti_expression() (*TM1py.ProcessService method*), 55
evaluate_ti_expression() (*TM1py.ProcessService method*), 55
execute() (*TM1py.ProcessService method*), 56
execute_audit_log_delta_request() (*TM1py.ServerService method*), 63
execute_chore() (*TM1py.ChoreService method*), 38
execute_mdx() (*TM1py.CellService method*), 14
execute_mdx() (*TM1py.DimensionService method*), 42
execute_mdx() (*TM1py.PowerBiService method*), 54
execute_mdx_cellcount() (*TM1py.CellService method*), 15
execute_mdx_csv() (*TM1py.CellService method*), 15
execute_mdx_dataframe() (*TM1py.CellService method*), 16
execute_mdx_dataframe_pivot() (*TM1py.CellService method*), 16
execute_mdx_dataframe_shaped() (*TM1py.CellService method*), 16
execute_mdx_elements_value_dict() (*TM1py.CellService method*), 17
execute_mdx_raw() (*TM1py.CellService method*), 17
execute_mdx_rows_and_values() (*TM1py.CellService method*), 18
execute_mdx_rows_and_values_string_set() (*TM1py.CellService method*), 18
execute_mdx_ui_array() (*TM1py.CellService method*), 18
execute_mdx_ui_dygraph() (*TM1py.CellService method*), 19
execute_mdx_values() (*TM1py.CellService method*), 20
execute_message_log_delta_request() (*TM1py.ServerService method*), 63
execute_process_with_return() (*TM1py.ProcessService method*), 56
execute_set_mdx() (*TM1py.ElementService method*), 44
execute_ti_code() (*TM1py.ProcessService method*), 56
execute_transaction_log_delta_request() (*TM1py.ServerService method*), 63
execute_unbound_process() (*TM1py.CellService method*), 20
execute_view() (*TM1py.CellService method*), 20
execute_view() (*TM1py.PowerBiService method*), 54
execute_view_cellcount() (*TM1py.CellService method*), 21
execute_view_csv() (*TM1py.CellService method*), 21
execute_view_dataframe() (*TM1py.CellService method*), 21
execute_view_dataframe_pivot() (*TM1py.CellService method*), 22
execute_view_dataframe_shaped() (*TM1py.CellService method*), 22
execute_view_elements_value_dict() (*TM1py.CellService method*), 23
execute_view_raw() (*TM1py.CellService method*), 23
execute_view_rows_and_values() (*TM1py.CellService method*), 24
execute_view_rows_and_values_string_set() (*TM1py.CellService method*), 24
execute_view_ui_array() (*TM1py.CellService method*), 25
execute_view_ui_dygraph() (*TM1py.CellService method*), 25
execute_view_values() (*TM1py.CellService

method), 26
execute_with_return() (TM1py.ProcessService method), 56
execution_mode (TM1py.Chore attribute), 71
exists() (TM1py.ApplicationService method), 11
exists() (TM1py.ChoreService method), 38
exists() (TM1py.CubeService method), 39
exists() (TM1py.DimensionService method), 42
exists() (TM1py.ElementService method), 45
exists() (TM1py.HierarchyService method), 51
exists() (TM1py.ProcessService method), 57
exists() (TM1py.SandboxService method), 60
exists() (TM1py.SubsetService method), 66
exists() (TM1py.ViewService method), 68
expression (TM1py.Subset attribute), 81
extract_cellset() (TM1py.CellService method), 27
extract_cellset_cellcount() (TM1py.CellService method), 27
extract_cellset_cells_raw() (TM1py.CellService method), 27
extract_cellset_composition() (TM1py.CellService method), 28
extract_cellset_csv() (TM1py.CellService method), 28
extract_cellset_csv_iter_json() (TM1py.CellService method), 28
extract_cellset_dataframe() (TM1py.CellService method), 29
extract_cellset_dataframe_pivot() (TM1py.CellService method), 29
extract_cellset_dataframe_shaped() (TM1py.CellService method), 29
extract_cellset_metadata_raw() (TM1py.CellService method), 30
extract_cellset_raw() (TM1py.CellService method), 30
extract_cellset_raw_response() (TM1py.CellService method), 30
extract_cellset_rows_and_values() (TM1py.CellService method), 31
extract_cellset_values() (TM1py.CellService method), 31

F

feeder_statements (TM1py.Rules attribute), 80
feedstrings (TM1py.Cube attribute), 72
feedstrings (TM1py.Rules attribute), 80
force (TM1py.Git attribute), 74
force (TM1py.GitPlan attribute), 74
format_string (TM1py.NativeView attribute), 77
frequency (TM1py.Chore attribute), 71
frequency_string (TM1py.ChoreFrequency attribute), 71
friendly_name (TM1py.User attribute), 81
from_dict() (TM1py.Chore class method), 71
from_dict() (TM1py.ChoreTask class method), 72
from_dict() (TM1py.Cube class method), 72
from_dict() (TM1py.Dimension class method), 73
from_dict() (TM1py.Element static method), 73
from_dict() (TM1py.ElementAttribute class method), 74
from_dict() (TM1py.Git class method), 74
from_dict() (TM1py.Hierarchy class method), 75
from_dict() (TM1py.MDXView class method), 76
from_dict() (TM1py.NativeView class method), 77
from_dict() (TM1py.Process class method), 79
from_dict() (TM1py.Sandbox class method), 80
from_dict() (TM1py.Subset class method), 81
from_dict() (TM1py.User class method), 81
from_json() (TM1py.Annotation class method), 70
from_json() (TM1py.Chore class method), 71
from_json() (TM1py.Cube class method), 72
from_json() (TM1py.Dimension class method), 73
from_json() (TM1py.ElementAttribute class method), 74
from_json() (TM1py.MDXView class method), 76
from_json() (TM1py.NativeView class method), 77
from_json() (TM1py.Process class method), 79
from_json() (TM1py.Sandbox class method), 80
from_json() (TM1py.Subset class method), 81
from_json() (TM1py.User class method), 81
from_string() (TM1py.ChoreFrequency class method), 71
from_string() (TM1py.ChoreStartTime class method), 72

G

generate_enable_sandbox_ti() (TM1py.CellService method), 32
get() (TM1py.AnnotationService method), 10
get() (TM1py.ApplicationService method), 11
get() (TM1py.ChoreService method), 38
get() (TM1py.CubeService method), 39
get() (TM1py.DimensionService method), 42
get() (TM1py.ElementService method), 45
get() (TM1py.HierarchyService method), 51
get() (TM1py.ProcessService method), 57
GET() (TM1py.RestService method), 59
get() (TM1py.SandboxService method), 60
get() (TM1py.SubsetService method), 66
get() (TM1py.ViewService method), 68
get_active_configuration() (TM1py.ServerService method), 63
get_active_session_threads() (TM1py.MonitoringService method), 53
get_active_threads() (TM1py.MonitoringService method), 53

get_active_users() (*TM1py.MonitoringService method*), 53
get_admin_host() (*TM1py.ServerService method*), 63
get_alias_element_attributes() (*TM1py.ElementService method*), 45
get_all() (*TM1py.AnnotationService method*), 10
get_all() (*TM1py.ChoreService method*), 38
get_all() (*TM1py.CubeService method*), 39
get_all() (*TM1py.ProcessService method*), 57
get_all() (*TM1py.SandboxService method*), 60
get_all() (*TM1py.ViewService method*), 68
get_all_element_identifiers() (*TM1py.ElementService method*), 45
get_all_groups() (*TM1py.SecurityService method*), 62
get_all_leaf_element_identifiers() (*TM1py.ElementService method*), 45
get_all_names() (*TM1py.ChoreService method*), 38
get_all_names() (*TM1py.CubeService method*), 39
get_all_names() (*TM1py.DimensionService method*), 42
get_all_names() (*TM1py.HierarchyService method*), 51
get_all_names() (*TM1py.ProcessService method*), 57
get_all_names() (*TM1py.SandboxService method*), 60
get_all_names() (*TM1py.SubsetService method*), 66
get_all_names() (*TM1py.ViewService method*), 68
get_all_names_with_rules() (*TM1py.CubeService method*), 40
get_all_names_without_rules() (*TM1py.CubeService method*), 40
get_all_user_names() (*TM1py.SecurityService method*), 62
get_all_users() (*TM1py.SecurityService method*), 62
get_ancestors() (*TM1py.Hierarchy method*), 75
get_attribute_of_elements() (*TM1py.ElementService method*), 45
get_audit_log_entries() (*TM1py.ServerService method*), 63
get_cellset_cells_count() (*TM1py.CellService method*), 32
get_configuration() (*TM1py.ServerService method*), 64
get_consolidated_element_names() (*TM1py.ElementService method*), 46
get_consolidated_elements() (*TM1py.ElementService method*), 46
get_control_cubes() (*TM1py.CubeService method*), 40
get_cube_service() (*TM1py.CellService method*), 32
get_current_user() (*TM1py.MonitoringService method*), 53
get_current_user() (*TM1py.SecurityService method*), 62
get_custom_security_groups() (*TM1py.SecurityService method*), 62
get_data_directory() (*TM1py.ServerService method*), 64
get_default_member() (*TM1py.HierarchyService method*), 52
get_descendant_edges() (*TM1py.Hierarchy method*), 75
get_descendants() (*TM1py.Hierarchy method*), 75
get_dimension_names() (*TM1py.CubeService method*), 40
get_dimension_names_for_writing() (*TM1py.CellService method*), 32
get_document() (*TM1py.ApplicationService method*), 11
get_edges() (*TM1py.ElementService method*), 46
get_element() (*TM1py.Hierarchy method*), 75
get_element_attribute_names() (*TM1py.ElementService method*), 46
get_element_attributes() (*TM1py.ElementService method*), 46
get_element_identifiers() (*TM1py.ElementService method*), 46
get_element_names() (*TM1py.ElementService method*), 46
get_element_names() (*TM1py.SubsetService method*), 66
get_element_principal_name() (*TM1py.ElementService method*), 47
get_element_service() (*TM1py.CellService method*), 32
get_element_types() (*TM1py.ElementService method*), 47
get_element_types_from_all_hierarchies() (*TM1py.ElementService method*), 47
get_elements() (*TM1py.ElementService method*), 47
get_elements_by_level() (*TM1py.ElementService method*), 47
get_elements_dataframe() (*TM1py.ElementService method*), 47
get_elements_filtered_by_attribute() (*TM1py.ElementService method*), 47
get_elements_filtered_by_wildcard() (*TM1py.ElementService method*), 48
get_elements_from_all_measure_hierarchies() (*TM1py.CellService method*), 32
get_error_log_file_content()

(*TM1py.CellService method*), 32
get_error_log_file_content() (*TM1py.ProcessService method*), 57
get_groups() (*TM1py.SecurityService method*), 62
get_hierarchy() (*TM1py.Dimension method*), 73
get_hierarchy_summary() (*TM1py.HierarchyService method*), 52
get_http_header() (*TM1py.RestService method*), 59
get_last_data_update() (*TM1py.CubeService method*), 40
get_last_message_from_processerrorlog() (*TM1py.ProcessService method*), 57
get_last_process_message_from_messagelogs() (*TM1py.ServerService method*), 64
get_leaf_element_names() (*TM1py.ElementService method*), 48
get_leaf_elements() (*TM1py.ElementService method*), 48
get_leaves_under_consolidation() (*TM1py.ElementService method*), 48
get_level_names() (*TM1py.ElementService method*), 48
get_levels_count() (*TM1py.ElementService method*), 48
get_mdx_view() (*TM1py.ViewService method*), 68
get_measure_dimension() (*TM1py.CubeService method*), 40
get_member_properties() (*TM1py.PowerBiService method*), 54
get_members_under_consolidation() (*TM1py.ElementService method*), 48
get_message_log_entries() (*TM1py.ServerService method*), 64
get_model_cubes() (*TM1py.CubeService method*), 40
get_monitoring_service() (*TM1py.RestService method*), 59
get_native_view() (*TM1py.ViewService method*), 69
get_number_of_consolidated_elements() (*TM1py.ElementService method*), 49
get_number_of_cubes() (*TM1py.CubeService method*), 40
get_number_of_dimensions() (*TM1py.DimensionService method*), 43
get_number_of_elements() (*TM1py.ElementService method*), 49
get_number_of_leaf_elements() (*TM1py.ElementService method*), 49
get_number_of_numeric_elements() (*TM1py.ElementService method*), 49
get_number_of_string_elements() (*TM1py.ElementService method*), 49
get_numeric_element_names() (*TM1py.ElementService method*), 49
get_numeric_elements() (*TM1py.ElementService method*), 49
get_parents() (*TM1py.ElementService method*), 49
get_parents_of_all_elements() (*TM1py.ElementService method*), 49
get_process_service() (*TM1py.ElementService method*), 49
get_processerrorlogs() (*TM1py.ProcessService method*), 57
get_product_version() (*TM1py.ServerService method*), 64
get_random_intersection() (*TM1py.CubeService method*), 40
get_read_only_users() (*TM1py.SecurityService method*), 62
get_server_name() (*TM1py.ServerService method*), 64
get_sessions() (*TM1py.MonitoringService method*), 53
get_static_configuration() (*TM1py.ServerService method*), 64
get_storage_dimension_order() (*TM1py.CubeService method*), 40
get_string_element_names() (*TM1py.ElementService method*), 49
get_string_elements() (*TM1py.ElementService method*), 49
get_threads() (*TM1py.MonitoringService method*), 53
get_transaction_log_entries() (*TM1py.ServerService method*), 64
get_user() (*TM1py.SecurityService method*), 62
get_user_names_from_group() (*TM1py.SecurityService method*), 62
get_users_from_group() (*TM1py.SecurityService method*), 63
get_value() (*TM1py.CellService method*), 32
get_view_content() (*TM1py.CellService method*), 32
Git (*class in TM1py*), 74
git_execute_plan() (*TM1py.GitService method*), 49
git_get_plans() (*TM1py.GitService method*), 49
git_init() (*TM1py.GitService method*), 49
git_pull() (*TM1py.GitService method*), 50
git_push() (*TM1py.GitService method*), 50
git_status() (*TM1py.GitService method*), 50
git_uninit() (*TM1py.GitService method*), 50
GitCommit (*class in TM1py*), 74
GitPlan (*class in TM1py*), 74
GitRemote (*class in TM1py*), 74
GitService (*class in TM1py*), 49

group_exists() (*TM1py.SecurityService method*), 63
groups (*TM1py.User attribute*), 82

H

has_feeders (*TM1py.Rules attribute*), 80
has_rules (*TM1py.Cube attribute*), 72
has_security_access (*TM1py.Process attribute*), 79
headers (*TM1py.Exceptions.Exceptions.TM1pyRestException attribute*), 83
HEADERS (*TM1py.RestService attribute*), 59
hierarchies (*TM1py.Dimension attribute*), 73
Hierarchy (*class in TM1py*), 75
hierarchy_exists() (*TM1py.ElementService method*), 49
hierarchy_name (*TM1py.Subset attribute*), 81
hierarchy_name (*TM1py.ViewAxisSelection attribute*), 82
hierarchy_name (*TM1py.ViewTitleSelection attribute*), 82
hierarchy_names (*TM1py.Dimension attribute*), 73
HierarchyService (*class in TM1py*), 50
hours (*TM1py.ChoreFrequency attribute*), 71

I

id (*TM1py.Annotation attribute*), 70
include_in_sandbox_dimension (*TM1py.Sandbox attribute*), 80
index (*TM1py.Element attribute*), 73
init_analytics() (*TM1py.Rules method*), 80
initialize_audit_log_delta_requests() (*TM1py.ServerService method*), 65
initialize_message_log_delta_requests() (*TM1py.ServerService method*), 65
initialize_transaction_log_delta_requests() (*TM1py.ServerService method*), 65
is_admin (*TM1py.RestService attribute*), 59
is_admin (*TM1py.User attribute*), 82
is_balanced() (*TM1py.HierarchyService method*), 52
is_connected() (*TM1py.RestService method*), 59
is_dynamic (*TM1py.Subset attribute*), 81
is_mdx_view() (*TM1py.ViewService method*), 69
is_native_view() (*TM1py.ViewService method*), 69
is_static (*TM1py.Subset attribute*), 81

K

KEYWORDS (*TM1py.Rules attribute*), 80

L

last_updated (*TM1py.Annotation attribute*), 70
last_updated_by (*TM1py.Annotation attribute*), 70
load() (*TM1py.CubeService method*), 40
load() (*TM1py.SandboxService method*), 61
lock() (*TM1py.CubeService method*), 41
logout() (*TM1py.RestService method*), 59
logout() (*TM1py.TM1Service method*), 67

M

make_static() (*TM1py.SubsetService method*), 67
MAX_STATEMENTS (*TM1py.Process attribute*), 78
max_statements() (*TM1py.Process static method*), 79
MAX_STATEMENTS_POST_11_8_015 (*TM1py.Process attribute*), 78
MDX (*TM1py.MDXView attribute*), 76
mdx (*TM1py.MDXView attribute*), 76
MDX (*TM1py.NativeView attribute*), 76
mdx (*TM1py.NativeView attribute*), 77
mdx (*TM1py.View attribute*), 82
MDXView (*class in TM1py*), 76
merge() (*TM1py.SandboxService method*), 61
metadata_procedure (*TM1py.Process attribute*), 79
minutes (*TM1py.ChoreFrequency attribute*), 71
MonitoringService (*class in TM1py*), 53
move() (*TM1py.Annotation method*), 70
MULTIPLE_COMMIT (*TM1py.Chore attribute*), 71

N

name (*TM1py.Chore attribute*), 71
name (*TM1py.Cube attribute*), 72
name (*TM1py.Dimension attribute*), 73
name (*TM1py.Element attribute*), 73
name (*TM1py.ElementAttribute attribute*), 74
name (*TM1py.Hierarchy attribute*), 75
name (*TM1py.Process attribute*), 80
name (*TM1py.Sandbox attribute*), 80
name (*TM1py.Subset attribute*), 81
name (*TM1py.User attribute*), 82
name (*TM1py.View attribute*), 82
NativeView (*class in TM1py*), 76
NUMERIC (*TM1py.Element.Types attribute*), 73
NUMERIC (*TM1py.ElementAttribute.Types attribute*), 74

O

object_name (*TM1py.Annotation attribute*), 70

P

parameters (*TM1py.ChoreTask attribute*), 72
parameters (*TM1py.Process attribute*), 80
password (*TM1py.User attribute*), 82
PATCH() (*TM1py.RestService method*), 59
plan_id (*TM1py.GitPlan attribute*), 74
POST() (*TM1py.RestService method*), 59
PowerBiService (*class in TM1py*), 54

Process (*class in TM1py*), 78
process_name (*TM1py.ChoreTask attribute*), 72
ProcessService (*class in TM1py*), 54
prolog_procedure (*TM1py.Process attribute*), 80
publish() (*TM1py.SandboxService method*), 61
PUT() (*TM1py.RestService method*), 59

R

re_authenticate() (*TM1py.TM1Service method*), 67
re_connect() (*TM1py.TM1Service method*), 67
reason (*TM1py.Exceptions.Exceptions.TM1pyRestException attribute*), 83
relative_proportional_spread() (*TM1py.CellService method*), 33
remote (*TM1py.Git attribute*), 74
remove_all_edges() (*TM1py.Hierarchy method*), 75
remove_all_edges() (*TM1py.HierarchyService method*), 52
remove_all_elements() (*TM1py.Hierarchy method*), 75
remove_column() (*TM1py.NativeView method*), 77
remove_edge() (*TM1py.ElementService method*), 49
remove_edge() (*TM1py.Hierarchy method*), 76
remove_edges() (*TM1py.Hierarchy method*), 76
remove_edges_related_to_element() (*TM1py.Hierarchy method*), 76
remove_edges_under_consolidation() (*TM1py.HierarchyService method*), 52
remove_element() (*TM1py.Hierarchy method*), 76
remove_element_attribute() (*TM1py.Hierarchy method*), 76
remove_group() (*TM1py.User method*), 82
remove_hierarchy() (*TM1py.Dimension method*), 73
remove_http_header() (*TM1py.RestService method*), 59
remove_parameter() (*TM1py.Process method*), 80
remove_row() (*TM1py.NativeView method*), 77
remove_title() (*TM1py.NativeView method*), 78
remove_user_from_group() (*TM1py.SecurityService method*), 63
remove_variable() (*TM1py.Process method*), 80
rename() (*TM1py.ApplicationService method*), 11
reschedule() (*TM1py.Chore method*), 71
reset() (*TM1py.SandboxService method*), 61
response (*TM1py.Exceptions.Exceptions.TM1pyRestException attribute*), 83
restore_from_file() (*TM1py.TM1Service class method*), 67
RestService (*class in TM1py*), 58
retrieve_async_response() (*TM1py.RestService method*), 59
rows (*TM1py.NativeView attribute*), 78
rule_statements (*TM1py.Rules attribute*), 80
Rules (*class in TM1py*), 80
rules (*TM1py.Cube attribute*), 72
rules_analytics (*TM1py.Rules attribute*), 80

S

Sandbox (*class in TM1py*), 80
sandbox_exists() (*TM1py.CellService method*), 33
sandboxing_disabled (*TM1py.RestService attribute*), 60
SandboxService (*class in TM1py*), 60
save_data() (*TM1py.ServerService method*), 65
save_to_file() (*TM1py.TM1Service method*), 67
schedule (*module*), 9
search_for_dimension() (*TM1py.CubeService method*), 41
search_for_dimension_substring() (*TM1py.CubeService method*), 41
search_for_parameter_value() (*TM1py.ChoreService method*), 38
search_for_process_name() (*TM1py.ChoreService method*), 38
search_for_rule_substring() (*TM1py.CubeService method*), 41
search_string_in_code() (*TM1py.ProcessService method*), 57
search_string_in_name() (*TM1py.ProcessService method*), 58
search_subset_in_native_views() (*TM1py.ViewService method*), 69
seconds (*TM1py.ChoreFrequency attribute*), 71
security_refresh() (*TM1py.SecurityService method*), 63
SecurityService (*class in TM1py*), 61
selected (*TM1py.ViewTitleSelection attribute*), 82
Server (*class in TM1py*), 80
ServerService (*class in TM1py*), 63
session_id (*TM1py.RestService attribute*), 60
set_local_start_time() (*TM1py.ChoreService method*), 38
set_time() (*TM1py.ChoreStartTime method*), 72
set_version() (*TM1py.RestService method*), 60
SINGLE_COMMIT (*TM1py.Chore attribute*), 71
skipcheck (*TM1py.Cube attribute*), 72
skipcheck (*TM1py.Rules attribute*), 80
start_performance_monitor() (*TM1py.ServerService method*), 65
start_time (*TM1py.Chore attribute*), 71
start_time_string (*TM1py.ChoreStartTime attribute*), 72
status_code (*TM1py.Exceptions.Exceptions.TM1pyRestException attribute*), 83
step (*TM1py.ChoreTask attribute*), 72

stop_performance_monitor() (*TM1py.ServerService method*), 65
STRING (*TM1py.Element.Types attribute*), 73
STRING (*TM1py.ElementAttribute.Types attribute*), 74
Subset (*class in TM1py*), 80
subset (*TM1py.ViewAxisSelection attribute*), 82
subset (*TM1py.ViewTitleSelection attribute*), 82
subsets (*TM1py.Hierarchy attribute*), 76
SubsetService (*class in TM1py*), 65
substitute_title() (*TM1py.MDXView method*), 76
substitute_title() (*TM1py.NativeView method*), 78
subtract() (*TM1py.ChoreStartTime method*), 72
summary (*TM1py.GitCommit attribute*), 74
suppress_empty_cells (*TM1py.NativeView attribute*), 78
suppress_empty_columns (*TM1py.NativeView attribute*), 78
suppress_empty_rows (*TM1py.NativeView attribute*), 78

T

tags (*TM1py.GitRemote attribute*), 75
tasks (*TM1py.Chore attribute*), 71
TCP_SOCKET_OPTIONS (*TM1py.RestService attribute*), 59
text (*TM1py.Annotation attribute*), 70
text (*TM1py.Rules attribute*), 80
titles (*TM1py.NativeView attribute*), 78
tm1project_delete() (*TM1py.GitService method*), 50
tm1project_get() (*TM1py.GitService method*), 50
tm1project_put() (*TM1py.GitService method*), 50
TM1pyException (*class in TM1py.Exceptions.Exceptions*), 83
TM1pyNotAdminException (*class in TM1py.Exceptions.Exceptions*), 82
TM1pyRestException (*class in TM1py.Exceptions.Exceptions*), 82
TM1pyTimeout (*class in TM1py.Exceptions.Exceptions*), 82
TM1pyVersionException (*class in TM1py.Exceptions.Exceptions*), 82
TM1pyWriteFailureException (*class in TM1py.Exceptions.Exceptions*), 83
TM1pyWritePartialFailureException (*class in TM1py.Exceptions.Exceptions*), 83
TM1Service (*class in TM1py*), 67
trace_cell_calculation() (*TM1py.CellService method*), 33
trace_cell_feeders() (*TM1py.CellService method*), 33
transaction_log_is_active() (*TM1py.CellService method*), 34
translate_to_boolean() (*TM1py.RestService static method*), 60
type (*TM1py.Subset attribute*), 81

U

undefvals (*TM1py.Cube attribute*), 73
undefvals (*TM1py.Rules attribute*), 80
undo_changeset() (*TM1py.CellService method*), 34
unique_name (*TM1py.Dimension attribute*), 73
unique_name (*TM1py.Element attribute*), 73
unload() (*TM1py.CubeService method*), 41
unload() (*TM1py.SandboxService method*), 61
unlock() (*TM1py.CubeService method*), 41
update() (*TM1py.AnnotationService method*), 10
update() (*TM1py.ApplicationService method*), 11
update() (*TM1py.ChoreService method*), 39
update() (*TM1py.CubeService method*), 41
update() (*TM1py.DimensionService method*), 43
update() (*TM1py.ElementService method*), 49
update() (*TM1py.HierarchyService method*), 52
update() (*TM1py.ProcessService method*), 58
update() (*TM1py.SandboxService method*), 61
update() (*TM1py.SubsetService method*), 67
update() (*TM1py.ViewService method*), 69
update_cellset() (*TM1py.CellService method*), 34
update_default_member() (*TM1py.HierarchyService method*), 52
update_document_from_file() (*TM1py.ApplicationService method*), 12
update_edge() (*TM1py.Hierarchy method*), 76
update_element() (*TM1py.Hierarchy method*), 76
update_element_attributes() (*TM1py.HierarchyService method*), 53
update_or_create() (*TM1py.ChoreService method*), 39
update_or_create() (*TM1py.CubeService method*), 42
update_or_create() (*TM1py.DimensionService method*), 43
update_or_create() (*TM1py.HierarchyService method*), 53
update_or_create() (*TM1py.ProcessService method*), 58
update_or_create() (*TM1py.SubsetService method*), 67
update_or_create() (*TM1py.ViewService method*), 69
update_or_create_document_from_file() (*TM1py.ApplicationService method*), 12
update_static_configuration() (*TM1py.ServerService method*), 65

update_storage_dimension_order() (*TM1py.CubeService method*), 42
update_user() (*TM1py.SecurityService method*), 63
update_user_password() (*TM1py.SecurityService method*), 63
url (*TM1py.Git attribute*), 74
urllib3_response_from_bytes() (*TM1py.RestService static method*), 60
User (*class in TM1py*), 81
user_exists() (*TM1py.SecurityService method*), 63
user_is_active() (*TM1py.MonitoringService method*), 54
user_type (*TM1py.User attribute*), 82
uses_alternate_hierarchies() (*TM1py.DimensionService method*), 43
utc_localize_time() (*TM1py.ServerService static method*), 65

V

variables (*TM1py.Process attribute*), 80
verify_response() (*TM1py.RestService static method*), 60
version (*TM1py.RestService attribute*), 60
version (*TM1py.TM1Service attribute*), 67
View (*class in TM1py*), 82
ViewAxisSelection (*class in TM1py*), 82
ViewService (*class in TM1py*), 67
ViewTitleSelection (*class in TM1py*), 82

W

wait_time_generator() (*TM1py.RestService static method*), 60
whoami (*TM1py.TM1Service attribute*), 67
write() (*TM1py.CellService method*), 34
write_async() (*TM1py.CellService method*), 35
write_dataframe() (*TM1py.CellService method*), 35
write_dataframe_async() (*TM1py.CellService method*), 36
write_through_blob() (*TM1py.CellService method*), 36
write_through_cellset() (*TM1py.CellService method*), 36
write_through_unbound_process() (*TM1py.CellService method*), 36
write_to_message_log() (*TM1py.ServerService method*), 65
write_value() (*TM1py.CellService method*), 37
write_values() (*TM1py.CellService method*), 37
write_values_through_cellset() (*TM1py.CellService method*), 37

Z

zfill_two() (*TM1py.ChoreService static method*), 39