PyTan Documentation

Release 1.0.0

Jim Olsen

December 05, 2014

1 Table of Contents **1**
- 1.1 Description . . . 1
- 1.2 Why it was created . . . 1
- 1.3 Requirements . . . 1
- 1.4 Installation . . . 2
- 1.5 pytan package . . . 2
- 1.6 taniumpy package . . . 41
- 1.7 xmltodict module . . . 55
- 1.8 ddt module . . . 56
- 1.9 threaded_http module . . . 57

2 Indices and tables **59**

Python Module Index **61**

Index **63**

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Description

This is a set of packages and scripts that provides a simple way for programmatically interfacing with Tanium's SOAP API. It is comprised of four parts:

- Tanium Server SOAP API: The SOAP server embedded into the Tanium server itself, listens on port 444 but is also available via port 443.
- TaniumPy Python Package (`taniumpy`): A python package comprised of a set of python objects automatically generated from the WSDL file that describe the Tanium SOAP API. These python objects handle the serialization and deserialization of XML to and from the Tanium Server SOAP API. Located in `lib/taniumpy`
- PyTan Python Package: (`pytan`): A python package that provides a set of methods to make interfacing with TaniumPy more human friendly. Located in `lib/pytan`
- PyTan Command Line Scripts: A set of command line scripts that utilize the PyTan Package (`pytan`) to make it easy for non-programmers to create/get/delete/ask/deploy objects via the Tanium Server SOAP API.

1.2 Why it was created

This was created to solve for the following needs:

- Create a python module (`taniumpy`) to provide a set of methods for making it easier to programmatically interface with Tanium via the SOAP API.
- Create a set of command line scripts utilizing the `taniumpy` module that handle the argument parsing, thereby providing non-programmers with command line access to the functionality therein.
- Provide a way to ask questions and get results via Python and/or the command line.
- Provide a way to deploy actions and get results via Python and/or the command line.
- Provide a way to export/import objects in JSON via Python and/or the command line.

1.3 Requirements

- Python 2.7
- A working install of Tanium Server 6.2

1.4 Installation

Windows Installation

- Download Python 2.7 from https://www.python.org/downloads/windows/
- Install Python 2.7 – if you accept the default paths it will install to `C:\Python27`
- Copy PyTan from github to your local machine somewhere
- If you did not accept the default install path for Python 2.7, edit `pytan\winbin\CONFIG.bat` to change the *PYTHON* variable to point to the full path of *python.exe*

OS X Installation

- OS X 10.8 and higher come with Python 2.7 out of the box
- Copy PyTan from github to your local machine somewhere

Linux Installation

- Ensure Python 2.7 is installed
- Ensure the first *python* binary in your path points to your Python 2.7 installation
- Copy PyTan from github to your local machine somewhere

1.5 pytan package

A python package that makes using (`taniumpy`) more human friendly.

`pytan.__version__` = **'1.0.0'**
Version of PyTan

`pytan.__copyright__` = **'Copyright 2014 Tanium'**
Copyright for PyTan

`pytan.__license__` = **'MIT'**
License for PyTan

`pytan.__author__` = **'Jim Olsen <jim.olsen@tanium.com>'**
Author of Pytan

1.5.1 pytan.handler module

The main `pytan` module that provides methods for programmatic use.

Handler Class

class `pytan.handler.Handler`(*username*, *password*, *host*, *port='444'*, *loglevel=0*, *debugformat=False*, ***kwargs*)

Bases: `object`

Creates a connection to a Tanium SOAP Server on host:port

Parameters **username** : str

username to connect to *host* with

password : str

password to connect to *host* with

host : str

hostname or ip of Tanium SOAP Server

port : int, optional

port of Tanium SOAP Server on *host*

loglevel : int, optional

0 should not print anything, 1 and higher will print more

debugformat : bool, optional

False use one line logformat, True use two lines

See also:

`pytan.constants.LOG_LEVEL_MAPS` maps a given *loglevel* to respective logger names and their logger levels

`pytan.constants.INFO_FORMAT` debugformat=False

`pytan.constants.DEBUG_FORMAT` debugformat=True

Notes

- port 444 is the default SOAP port
- port 443 forwards /soap/ URLs to the SOAP port
- Use port 444 if you have direct access to it

Example: Create a Handler object

Setup a Handler() object:

```
>>> import sys
>>> sys.path.append('/path/to/pytan/')
>>> import pytan
>>> handler = pytan.Handler('username', 'password', 'host')
```

Handler Methods: Questions and Actions

Ask a Question

`Handler.ask`(***kwargs*)

Ask a type of question and get the results back

Parameters **qtype** : str

type of question to ask: saved_question, manual, or manual_human

Returns **result** : dict, containing:

- *question_object* : one of the following depending on *qtype*: taniumpy.object_types.question.Question or taniumpy.object_types.saved_question.SavedQuestion
- *question_results* : taniumpy.object_types.result_set.ResultSet

See also:

pytan.constants.Q_OBJ_MAP maps qtype to a method in Handler()

Ask a Saved Question

Handler.**ask_saved**(***kwargs*)

Ask a saved question and get the results back

Parameters **id** : int, list of int, optional

id of saved question to ask

name : str, list of str

name of saved question

Returns **ret** : dict, containing

- *question_object* : taniumpy.object_types.saved_question.SavedQuestion
- *question_results* : taniumpy.object_types.result_set.ResultSet

See also:

pytan.constants.ASK_KWARGS list of kwargs that can be passed to taniumpy.question_asker.QuestionAsker

Notes

id or name must be supplied

Asking a Manual Question

Handler.**ask_manual**(*get_results=True*, ***kwargs*)

Ask a manual question using definitions and get the results back

This method requires in-depth knowledge of how filters and options are created in the API, and as such is not meant for human consumption. Use ask_manual_human() instead.

Parameters **sensor_defs** : str, dict, list of str or dict

sensor definitions

question_filter_defs : dict, list of dict, optional

question filter definitions

question_option_defs : dict, list of dict, optional

question option definitions

get_results : bool, optional

- True: wait for result completion after asking question
- False: just ask the question and return it in *ret*

Returns **ret** : dict, containing:

- *question_object* : `taniumpy.object_types.question.Question`
- *question_results* : `taniumpy.object_types.result_set.ResultSet`

See also:

`pytan.constants.FILTER_MAPS` valid filter dictionaries for filters

`pytan.constants.OPTION_MAPS` valid option dictionaries for options

`pytan.constants.ASK_KWARGS` list of kwargs that can be passed to `taniumpy.question_asker.QuestionAsker`

Examples

```
>>> # example of str for sensor_defs
>>> sensor_defs = 'Sensor1'
```

```
>>> # example of dict for sensor_defs
>>> sensor_defs = {
... 'name': 'Sensor1',
...     'filter': {
...         'operator': 'RegexMatch',
...         'not_flag': 0,
...         'value': '.*'
...     },
...     'params': {'key': 'value'},
...     'options': {'and_flag': 1}
... }
```

```
>>> # example of dict for question_filter_defs
>>> question_filter_defs = {
...     'operator': 'RegexMatch',
...     'not_flag': 0,
...     'value': '.*'
... }
```

`Handler.`**`ask_manual_human`**`(**kwargs)`

Ask a manual question using human strings and get the results back

This method takes a string or list of strings and parses them into their corresponding definitions needed by `ask_manual()`

Parameters **sensors** : str, list of str

sensors (columns) to include in question

question_filters : str, list of str, optional

filters that apply to the whole question

question_options : str, list of str, optional

options that apply to the whole question

get_results : bool, optional

True: wait for result completion after asking question False: just ask the question and return it in result

Returns **result** : dict, containing:

- *question_object* : `taniumpy.object_types.question.Question`
- *question_results* : `taniumpy.object_types.result_set.ResultSet`

See also:

`pytan.constants.FILTER_MAPS` valid filter dictionaries for filters

`pytan.constants.OPTION_MAPS` valid option dictionaries for options

`pytan.constants.ASK_KWARGS` list of kwargs that can be passed to `taniumpy.question_asker.QuestionAsker`

Examples

```
>>> # example of str for `sensors`
>>> sensors = 'Sensor1'
```

```
>>> # example of str for `sensors` with params
>>> sensors = 'Sensor1{key:value}'
```

```
>>> # example of str for `sensors` with params and filter
>>> sensors = 'Sensor1{key:value}, that contains example text'
```

```
>>> # example of str for `sensors` with params and filter and options
>>> sensors = (
...     'Sensor1{key:value}, that contains example text,'
...     'opt:ignore_case, opt:max_data_age:60'
... )
```

```
>>> # example of str for question_filters
>>> question_filters = 'Sensor2, that contains example test'
```

```
>>> # example of list of str for question_options
>>> question_options = ['max_data_age:3600', 'and']
```

Deploy an Action

`Handler.`**`deploy_action`**(*run=False*, *get_results=True*, ***kwargs*)

Deploy an action and get the results back

This method requires in-depth knowledge of how filters and options are created in the API, and as such is not meant for human consumption. Use `deploy_action_human()` instead.

Parameters **package_def** : dict

definition that describes a package

action_filter_defs : str, dict, list of str or dict, optional

action filter definitions

action_option_defs : dict, list of dict, optional

action filter option definitions

start_seconds_from_now : int, optional

start action N seconds from now

expire_seconds : int, optional

expire action N seconds from now, will be derived from package if not supplied

run : bool, optional

- False: just ask the question that pertains to verify action, export the results to CSV, and raise RunFalse – does not deploy the action
- True: actually deploy the action

get_results : bool, optional

- True: wait for result completion after deploying action
- False: just deploy the action and return the object in *ret*

Returns **ret** : dict, containing:

- *action_object* : `taniumpy.object_types.action.Action`
- *action_results* : `taniumpy.object_types.result_set.ResultSet`
- *action_progress_human* : str, progress map in human form
- *action_progress_map* : dict, progress map in dictionary form
- *pre_action_question_results* : `taniumpy.object_types.result_set.ResultSet`

See also:

`pytan.constants.FILTER_MAPS` valid filter dictionaries for filters

`pytan.constants.OPTION_MAPS` valid option dictionaries for options

Examples

```
>>> # example of dict for `package_def`
>>> package_def = {'name': 'PackageName1', 'params':{'param1': 'value1'}}
```

```
>>> # example of str for `action_filter_defs`
>>> action_filter_defs = 'Sensor1'
```

```
>>> # example of dict for `action_filter_defs`
>>> action_filter_defs = {
... 'name': 'Sensor1',
...     'filter': {
...         'operator': 'RegexMatch',
...         'not_flag': 0,
...         'value': '.*'
...     },
```

```
...     'options': {'and_flag': 1}
... }
```

Handler.**deploy_action_human**(***kwargs*)

Deploy an action and get the results back

This method takes a string or list of strings and parses them into their corresponding definitions needed by `deploy_action()`

Parameters **package** : str

each string must describe a package

action_filters : str, list of str, optional

each string must describe a sensor and a filter which limits which computers the action will deploy *package* to

action_options : str, list of str, optional

options to apply to *action_filters*

start_seconds_from_now : int, optional

start action N seconds from now

expire_seconds : int, optional

expire action N seconds from now, will be derived from package if not supplied

run : bool, optional

- False: just ask the question that pertains to verify action, export the results to CSV, and raise RunFalse – does not deploy the action
- True: actually deploy the action

get_results : bool, optional

- True: wait for result completion after deploying action
- False: just deploy the action and return the object in *ret*

Returns **ret** : dict, containing:

- *action_object* : `taniumpy.object_types.action.Action`
- *action_results* : `taniumpy.object_types.result_set.ResultSet`
- *action_progress_human* : str, progress map in human form
- *action_progress_map* : dict, progress map in dictionary form
- *pre_action_question_results* : `taniumpy.object_types.result_set.ResultSet`

See also:

pytan.constants.FILTER_MAPS valid filter dictionaries for filters

pytan.constants.OPTION_MAPS valid option dictionaries for options

Examples

```
>>> # example of str for `package`
>>> package = 'Package1'
```

```
>>> # example of str for `package` with params
>>> package = 'Package1{key:value}'
```

```
>>> # example of str for `action_filters` with params and filter for sensors
>>> action_filters = 'Sensor1{key:value}, that contains example text'
```

```
>>> # example of list of str for `action_options`
>>> action_options = ['max_data_age:3600', 'and']
```

Handler.**deploy_action_asker**(*action_id*, *passed_count=0*)

Checks the results of a deploy action job and waits for completion

Parameters **action_id** : int

id of deploy action to get results for and wait on completion

passed_count : int, optional

the number of servers that must equate "completed" in order for deploy action to be recognized as completed

Returns **ret** : dict, containing:

- *action_object* : taniumpy.object_types.action.Action
- *action_results* : taniumpy.object_types.result_set.ResultSet
- *action_progress_human* : str, progress map in human form
- *action_progress_map* : dict, progress map in dictionary form

See also:

pytan.constants.ACTION_RESULT_STATUS maps the values in *Action Statuses* columns to success/completed/failed/etc

Stopping an Action

Handler.**stop_action**(*id*, ***kwargs*)

Stop an action

Parameters **id** : int

id of action to stop

Returns **action_stop_obj** : taniumpy.object_types.action_stop.ActionStop

The object containing the ID of the action stop job

Handler Methods: Exporting/Importing Objects

Import an API Object from JSON

Handler.**create_from_json**(*objtype*, *json_file*)
Creates a new object using the SOAP api from a json file

Parameters **objtype** : str

Type of object described in *json_file*

json_file : str

path to JSON file that describes an API object

Returns **ret** : taniumpy.object_types.base.BaseType

TaniumPy object added to Tanium SOAP Server

See also:

pytan.constants.GET_OBJ_MAP maps objtype to supported 'create_json' types

Load a Python Object from JSON
Handler.**load_taniumpy_from_json**(*json_file*)
Opens a json file and parses it into an taniumpy object

Parameters **json_file** : str

path to JSON file that describes an API object

Returns **obj** : taniumpy.object_types.base.BaseType

TaniumPy object converted from json file

Export Object
Handler.**export_obj**(*obj*, *export_format*, ***kwargs*)
Exports a python API object to a given export format

Parameters **obj** : taniumpy.object_types.base.BaseType or taniumpy.object_types.result_set.ResultSet

TaniumPy object to export

export_format : str

the number of servers that must equate "completed" in order for deploy action to be recognized as completed

header_sort : list of str, bool, optional

- for *export_format* csv and *obj* types taniumpy.object_types.base.BaseType or taniumpy.object_types.result_set.ResultSet
- True: sort the headers automatically
- False: do not sort the headers at all
- list of str: sort the headers returned by priority based on provided list

header_add_sensor : bool, optional

- for *export_format* csv and *obj* type taniumpy.object_types.result_set.ResultSet
- False: do not prefix the headers with the associated sensor name for each column
- True: prefix the headers with the associated sensor name for each column

header_add_type : bool, optional

- for *export_format* csv and *obj* type taniumpy.object_types.result_set.ResultSet
- False: do not postfix the headers with the result type for each column
- True: postfix the headers with the result type for each column

expand_grouped_columns : bool, optional

- for *export_format* csv and *obj* type taniumpy.object_types.result_set.ResultSet
- False: do not expand multiline row entries into their own rows
- True: expand multiline row entries into their own rows

explode_json_string_values : bool, optional

- for *export_format* json or csv and *obj* type taniumpy.object_types.base.BaseType
- False: do not explode JSON strings in object attributes into their own object attributes
- True: explode JSON strings in object attributes into their own object attributes

minimal : bool, optional

- for *export_format* xml and *obj* type taniumpy.object_types.base.BaseType
- False: include empty attributes in XML output
- True: do not include empty attributes in XML output

Returns **result** : str

the contents of exporting *export_format*

See also:

pytan.constants.EXPORT_MAPS maps the type *obj* to *export_format* and the optional args supported for each

Export Object to Report File

Handler.**export_to_report_file**(*obj*, *export_format*, ***kwargs*)

Exports a python API object to a file

Parameters **obj** : taniumpy.object_types.base.BaseType or taniumpy.object_types.result_set.ResultSet

TaniumPy object to export

export_format : str

the number of servers that must equate "completed" in order for deploy action to be recognized as completed

header_sort : list of str, bool, optional

- for *export_format* csv and *obj* types taniumpy.object_types.base.BaseType or taniumpy.object_types.result_set.ResultSet
- True: sort the headers automatically
- False: do not sort the headers at all
- list of str: sort the headers returned by priority based on provided list

header_add_sensor : bool, optional

- for *export_format* csv and *obj* type taniumpy.object_types.result_set.ResultSet

- False: do not prefix the headers with the associated sensor name for each column
- True: prefix the headers with the associated sensor name for each column

header_add_type : bool, optional

- for *export_format* csv and *obj* type `taniumpy.object_types.result_set.ResultSet`
- False: do not postfix the headers with the result type for each column
- True: postfix the headers with the result type for each column

expand_grouped_columns : bool, optional

- for *export_format* csv and *obj* type `taniumpy.object_types.result_set.ResultSet`
- False: do not expand multiline row entries into their own rows
- True: expand multiline row entries into their own rows

explode_json_string_values : bool, optional

- for *export_format* json or csv and *obj* type `taniumpy.object_types.base.BaseType`
- False: do not explode JSON strings in object attributes into their own object attributes
- True: explode JSON strings in object attributes into their own object attributes

minimal : bool, optional

- for *export_format* xml and *obj* type `taniumpy.object_types.base.BaseType`
- False: include empty attributes in XML output
- True: do not include empty attributes in XML output

report_file: str, optional

filename to save report as, will be automatically generated if not supplied

report_dir: str, optional

directory to save report in, if not supplied, will be extracted from *report_file*. if no directory in *report_file* or *report_file* not specified, will use current working directory.

prefix: str, optional

prefix to add to *report_file*

postfix: str, optional

postfix to add to *report_file*

Returns **report_path** : str

the full path to the file created with contents of *result*

result : str

the str of *export_format*

Handler Methods: Creating Objects

Create a Group

`Handler.create_group`(*groupname*, *filters=[]*, *filter_options=[]*)

Create a group object

Parameters **groupname** : str

name of group to create

filters : str or list of str, optional

each string must describe a filter

filter_options : str or list of str, optional

each string must describe an option for *filters*

Returns **group_obj** : `taniumpy.object_types.group.Group`

TaniumPy object added to Tanium SOAP Server

See also:

`pytan.constants.FILTER_MAPS` valid filters for filters

`pytan.constants.OPTION_MAPS` valid options for filter_options

Create a Package

`Handler.create_package`(*name*, *command*, *display_name=''*, *file_urls=[]*, *command_timeout_seconds=600*, *expire_seconds=600*, *parameters_json_file=''*, *verify_filters=[]*, *verify_filter_options=[]*, *verify_expire_seconds=600*)

Create a package object

Parameters **name** : str

name of package to create

command : str

command to execute

display_name : str, optional

display name of package

file_urls : list of strings, optional

- URL of file to add to package
- can optionally define download_seconds by using SECONDS::URL
- can optionally define file name by using FILENAME||URL
- can combine optionals by using SECONDS::FILENAME||URL
- FILENAME will be extracted from basename of URL if not provided

command_timeout_seconds : int, optional

timeout for command execution in seconds

parameters_json_file : str, optional

path to json file describing parameters for package

expire_seconds : int, optional

timeout for action expiry in seconds

verify_filters : str or list of str, optional

each string must describe a filter to be used to verify the package

verify_filter_options : str or list of str, optional

each string must describe an option for *verify_filters*

verify_expire_seconds : int, optional

timeout for verify action expiry in seconds

Returns **package_obj** : `taniumpy.object_types.package_spec.PackageSpec`

TaniumPy object added to Tanium SOAP Server

See also:

`pytan.constants.FILTER_MAPS` valid filters for verify_filters

`pytan.constants.OPTION_MAPS` valid options for verify_filter_options

Create a Sensor

`Handler.create_sensor()`

Create a sensor object

Raises **HandlerError** : `pytan.utils.HandlerError`

Warning: Not currently supported, too complicated to add. Use `create_from_json()` instead for this object type!

Create a User

`Handler.create_user`(*username*, *rolename=[]*, *roleid=[]*, *properties=[]*)

Create a user object

Parameters **username** : str

name of user to create

rolename : str or list of str, optional

name(s) of roles to add to user

roleid : int or list of int, optional

id(s) of roles to add to user

properties: list of list of strs, optional

- each list must be a 2 item list:
- list item 1 property name
- list item 2 property value

Returns **user_obj** : `taniumpy.object_types.user.User`

TaniumPy object added to Tanium SOAP Server

Create a Whitelisted URL
`Handler.create_whitelisted_url(url, regex=False, download_seconds=86400, properties=[])`

Create a whitelisted url object

Parameters **url** : str

text of new url

regex : bool, optional

- True: *url* is a regex pattern
- False: *url* is not a regex pattern

download_seconds : int, optional

how often to re-download *url*

properties: list of list of strs, optional

- each list must be a 2 item list:
- list item 1 property name
- list item 2 property value

Returns **url_obj** : `taniumpy.object_types.white_listed_url.WhiteListedUrl`

TaniumPy object added to Tanium SOAP Server

Handler Methods: Deleting Objects

Delete an Object
`Handler.delete(objtype, **kwargs)`

Delete an object type

Parameters **objtype** : string

type of object to delete

id/name/hash : int or string, list of int or string

search attributes of object to delete, must supply at least one valid search attr

Returns **ret** : dict

dict containing deploy action object and results from deploy action

See also:

`pytan.constants.GET_OBJ_MAP` maps objtype to supported 'search' keys

Handler Methods: Getting Objects

Get Single or Multiple Objects of a type
`Handler.get(objtype, **kwargs)`

Get an object type

Parameters **objtype** : string

type of object to get

id/name/hash : int or string, list of int or string

search attributes of object to get, must supply at least one valid search attr

See also:

`pytan.constants.GET_OBJ_MAP` maps objtype to supported 'search' keys

Get All Objects of a type

`Handler.get_all(objtype, **kwargs)`

Get all objects of a type

Parameters **objtype** : string

type of object to get

See also:

`pytan.constants.GET_OBJ_MAP` maps objtype to supported 'search' keys

Handler Methods: Getting Result Data / Result Info

`Handler.get_result_data(obj, aggregate=False, **kwargs)`

Get the result data for a python API object

This method issues a GetResultData command to the SOAP api for *obj*. GetResultData returns the columns and rows that are currently available for *obj*.

Parameters **obj** : `taniumpy.object_types.base.BaseType`

object to get result data for

aggregate : bool, optional

- False: get all the data
- True: get just the aggregate data (row counts of matches)

Returns **rd** : `taniumpy.object_types.result_set.ResultSet`

The return of GetResultData for *obj*

`Handler.get_result_info(obj, **kwargs)`

Get the result info for a python API object

This method issues a GetResultInfo command to the SOAP api for *obj*. GetResultInfo returns information about how many servers have passed the *obj*, total number of servers, and so on.

Parameters **obj** : `taniumpy.object_types.base.BaseType`

object to get result data for

Returns **ri** : `taniumpy.object_types.result_info.ResultInfo`

The return of GetResultData for *obj*

Handler Methods: Private Methods

Handler.**_find**(*api_object*, ***kwargs*)
Wrapper for interfacing with taniumpy.session.Session.find()

Handler.**_get_multi**(*obj_map*, ***kwargs*)
Find multiple item wrapper using _find()

Handler.**_get_single**(*obj_map*, ***kwargs*)
Find single item wrapper using _find()

Handler.**_single_find**(*obj_map*, *k*, *v*, ***kwargs*)
Wrapper for single item searches interfacing with taniumpy.session.Session.find()

Handler.**_get_sensor_defs**(*defs*)
Uses get() to update a definition with a sensor object

Handler.**_get_package_def**(*d*)
Uses get() to update a definition with a package object

Handler.**_export_class_BaseType**(*obj*, *export_format*, ***kwargs*)
Handles exporting taniumpy.object_types.base.BaseType

Handler.**_export_class_ResultSet**(*obj*, *export_format*, ***kwargs*)
Handles exporting taniumpy.object_types.result_set.ResultSet

Handler.**_export_format_csv**(*obj*, ***kwargs*)
Handles exporting format: CSV

Handler.**_export_format_json**(*obj*, ***kwargs*)
Handles exporting format: JSON

Handler.**_export_format_xml**(*obj*, ***kwargs*)
Handles exporting format: XML

1.5.2 pytan.constants module

PyTan Constants

This contains a number of constants that drive PyTan.

pytan.constants.**ACTION_RESULT_STATUS = {'Verified.': ['no_verify_done', 'verify_done', 'verify_success'], 'Succeeded**
Maps a deploy action result status to it's respective end states.

pytan.constants.**ASK_KWARGS = ['timeout', 'polling_interval', 'pct_complete_threshold']**
A list of arguments that will be passed on to the question asker/poller taniumpy.question_asker.QuestionAsker

pytan.constants.**DEBUG_FORMAT = '[%(lineno)-5d - %(filename)20s:%(funcName)s()] %(asctime)s\n%(levelname)-8s %**
Logging format for debugformat=True

pytan.constants.**EXPORT_MAPS = {'ResultSet': {'json': [], 'csv': [{'valid_list_types': ['str', 'unicode'], 'key': 'header_sor**

Maps a given TaniumPy object to the list of supported export formats for each object type, and the valid optional argume

- key: the optional argument name itself
- valid_types: the valid python types that are allowed to be passed as a value to *key*
- valid_list_types: the valid python types in str format that are allowed to be passed in a list, if list is one of the *valid_types*

pytan.constants.**FILTER_MAPS = [{'operator': 'Less', 'not_flag': 0, 'human': ['<', 'less', 'lt']}, {'operator': 'Less', 'not_f**

Maps a given set of human strings into the various filter attributes used by the SOAP API. Also used to verify that a manu

- human: a list of human strings that can be used after '*, that*'. Ex: '*, that* `contains value`'
- operator: the filter operator used by the SOAP API when building a filter that matches *human*
- not_flag: the value to set on *not_flag* when building a filter that matches *human*
- pre_value: the prefix to add to the `value` when building a filter
- post_value: the postfix to add to the `value` when building a filter

pytan.constants.**FILTER_RE = ',\\s*that'**
The regex that is used to find filters in a string. Ex: *Sensor1, `that` contains blah*

pytan.constants.**GET_OBJ_MAP = {'user': {'search': ['id'], 'all': 'UserList', 'manual': True, 'multi': None, 'single': 'Use**

Maps an object type from a human friendly string into various aspects:

- single: The `TaniumPy` object used to find singular instances of this object type
- multi: The `TaniumPy` object used to find multiple instances of this object type
- all: The `TaniumPy` object used to find all instances of this object type
- search: The list of attributes that can be used with the Tanium SOAP API for searches
- manual: Whether or not this object type is allowed to do a manual search, that is – allow the user to specify an attribute that is not in search, which will get ALL objects of that type then search for a match based on attribute values for EVERY key/value pair supplied
- delete: Whether or not this object type can be deleted
- create_json: Whether or not this object type can be created by importing from JSON

pytan.constants.**INFO_FORMAT = '%(asctime)s %(levelname)-8s %(name)s: %(message)s'**
Logging format for debugformat=False

pytan.constants.**LOG_LEVEL_MAPS = [(0, {'api.session.http': 'WARN', 'api.session': 'WARN', 'handler': 'WARN', 'ques**

Map for loglevel(int) -> logger -> logger level(logging.INFO|WARN|DEBUG|...). Higher loglevels will include all levels up

- int, loglevel
- dict, *{{logger_name: logger_level}}* for this loglevel

pytan.constants.**OPTION_MAPS = [{'destination': 'filter', 'attrs': {'ignore_case_flag': 1}, 'human': 'ignore_case', 'valid_**

Maps a given human string into the various options for filters used by the SOAP API. Also used to verify that a manually

- human: the human string that can be used after '*opt:*'. Ex: '*opt*:`value_type`:`value`'
- destination: the type of object this option can be applied to (filter or group)
- attrs: the attributes and their values used by the SOAP API when building a filter with an option that matches *human*
- attr: the attribute used by the SOAP API when building a filter with an option that matches *human*. `value` is pulled from after a *:* when only attr exists for an option map, and not attrs.
- valid_values: if supplied, the list of valid values for this option
- valid_type: performs type checking on the value supplied to verify it is correct

- human_type: the human string for the value type if the option requires a value

pytan.constants.**OPTION_RE = ',\\s*opt:'**
The regex that is used to find options in a string. Ex: *Sensor1, that contains blah*, `opt:`*ignore_case*, `opt:`*max_data_age:3600*

pytan.constants.**PARAM_DELIM = '||'**
The string to surround a parameter with when passing parameters to the SOAP API for a sensor in a question. Ex: `||`*parameter_key*`||`

pytan.constants.**PARAM_KEY_SPLIT = '='**
The string that is used to split parameter key from parameter value. Ex: *key1=value1*

pytan.constants.**PARAM_RE = '\\{(.*?)\\}'**
The regex that is used to parse parameters from a human string. Ex: ala {key1=value1}

pytan.constants.**PARAM_SPLIT_RE = '(?<!\\\\),'**
The regex that is used to split multiple parameters. Ex: key1=value1, key2=value2

pytan.constants.**Q_OBJ_MAP = {'manual': {'handler': 'ask_manual'}, 'saved': {'handler': 'ask_saved'}, 'manual_human**
Maps a question type from a human friendly string into the handler method that supports each type

pytan.constants.**REQ_KWARGS = ['hide_errors_flag', 'include_answer_times_flag', 'row_counts_only_flag', 'aggregate_ov**
A list of arguments that will be pulled from any respective kwargs for most calls to `taniumpy.session.Session`

pytan.constants.**SELECTORS = ['id', 'name', 'hash']**
The search selectors that can be extracted from a string. Ex: `name`:*Sensor1,* or `id`:*1*, or `hash`:*1111111*

pytan.constants.**SENSOR_TYPE_MAP = {0: 'Hash', 1: 'String', 2: 'Version', 3: 'NumericDecimal', 4: 'BESDate', 5: 'IPAd**
Maps a Result type from the Tanium SOAP API from an int to a string

1.5.3 pytan.utils module

Collection of exceptions, classes, and methods used throughout `pytan`

Utility Classes: Exceptions

Exceptions used throughout `pytan`:

exception `pytan.utils.`**HandlerError**
Bases: `exceptions.Exception`

Exception thrown for most errors in `pytan.handler`

exception `pytan.utils.`**HumanParserError**
Bases: `exceptions.Exception`

Exception thrown for errors while parsing human strings from `pytan.handler`

exception `pytan.utils.`**DefinitionParserError**
Bases: `exceptions.Exception`

Exception thrown for errors while parsing definitions from `pytan.handler`

exception `pytan.utils.`**RunFalse**
Bases: `exceptions.Exception`

Exception thrown when run=False from `pytan.handler.Handler.deploy_action()`

Utility Classes: Logging handlers

class pytan.utils.**SplitStreamHandler**
Bases: logging.Handler

Custom logging.Handler class that sends all messages that are logging.INFO and below to STDOUT, and all messages that are logging.WARNING and above to STDERR

emit(*record*)

Utility Classes: Argument Parsers for Command Line Scripts

class pytan.utils.**CustomArgFormat**(*prog*, *indent_increment=2*, *max_help_position=24*, *width=None*)
Bases: argparse.ArgumentDefaultsHelpFormatter, argparse.RawDescriptionHelpFormatter

Multiple inheritance Formatter class for argparse.ArgumentParser.

If a argparse.ArgumentParser class uses this as it's Formatter class, it will show the defaults for each argument in the *help* output

class pytan.utils.**CustomArgParse**(**args*, ***kwargs*)
Bases: argparse.ArgumentParser

Custom argparse.ArgumentParser class which does a number of things:

- Uses pytan.utils.CustomArgFormat as it's Formatter class, if none was passed in
- Prints help if there is an error
- Prints the help for any subparsers that exist

error(*message*)

print_help(***kwargs*)

Utility Functions: Logging

pytan.utils.**change_console_format**(*debug=False*)
Changes the logging format for console handler to pytan.constants.DEBUG_FORMAT or pytan.constants.INFO_FORMAT

Parameters **debug** : bool, optional

- False : set logging format for console handler to pytan.constants.INFO_FORMAT
- True : set logging format for console handler to pytan.constants.DEBUG_FORMAT

pytan.utils.**remove_logging_handler**(*name*)
Removes a logging handler

Parameters **name** : str

name of logging handler to remove. if name == 'all' then all logging handlers are removed

pytan.utils.**set_all_loglevels**(*level='DEBUG'*)
Sets all loggers that the logging system knows about to a given logger level

pytan.utils.**set_log_levels**(*loglevel=0*)
Enables loggers based on loglevel and pytan.constants.LOG_LEVEL_MAPS

Parameters **loglevel** : int, optional

loglevel to match against each item in pytan.constants.LOG_LEVEL_MAPS - each item that is greater than or equal to loglevel will have the according loggers set to their respective levels identified there-in.

pytan.utils.**setup_console_logging**()
Creates a console logging handler using SplitStreamHandler

Utility Functions: Type Checking

pytan.utils.**is_dict**(*l*)
returns True if *l* is a dictionary, False if not

pytan.utils.**is_list**(*l*)
returns True if *l* is a list, False if not

pytan.utils.**is_num**(*l*)
returns True if *l* is a number, False if not

pytan.utils.**is_str**(*l*)
returns True if *l* is a string, False if not

Utility Functions: Misc

pytan.utils.**get_dict_list_items**(*d*, *i*)
Gets keys from dict *d* if any item in list *i* is in the list value for each key

Parameters **d** : dict of str

dict to get strs from if list contains any item from *i*

i : list of str

list of strs to check if for existence in any lists in *d*

Returns **list** : list of str

list of strings from *d* that have *i* in their values

pytan.utils.**get_dict_list_len**(*d*, *keys=[]*, *negate=False*)
Gets the sum of each list in dict *d*

Parameters **d** : dict of str

dict to sums of

keys : list of str

list of keys to get sums of, if empty gets a sum of all keys

negate : bool

- only used if keys supplied
- False : get the sums of *d* that do match keys
- True : get the sums of *d* that do not match keys

Returns **list_len** : int

sum of lists in *d* that match keys

pytan.utils.**get_now**()
Get current time in human friendly format

Returns str :

str of current time return from `human_time()`

pytan.utils.**human_time**(*t*, *tformat='%Y_%m_%d-%H_%M_%S-%Z'*)

Get time in human friendly format

Parameters **t** : int, float, time

either a unix epoch or struct_time object to convert to string

tformat : str, optional

format of string to convert time to

Returns str :

t converted to str

pytan.utils.**jsonify**(*v*, *indent=2*, *sort_keys=True*)

Turns python object *v* into a pretty printed JSON string

Parameters **v** : object

python object to convert to JSON

indent : int, 2

number of spaces to indent JSON string when pretty printing

sort_keys : bool, True

sort keys of JSON string when pretty printing

Returns str :

JSON pretty printed string

pytan.utils.**port_check**(*address*, *port*, *timeout=5*)

Check if *address*:*port* can be reached within *timeout*

Parameters **address** : str

hostname/ip address to check *port* on

port : int

port to check on *address*

timeout : int, optional

timeout after N seconds of not being able to connect

Returns `socket` or False :

if connection succeeds, the socket object is returned, else False is returned

pytan.utils.**seconds_from_now**(*secs=0*, *tz='utc'*)

Get time in Tanium SOAP API format *secs* from now

Parameters **secs** : int

seconds from now to get time str

tz : str, optional

time zone to return string in, default is 'utc' - supplying anything else will supply local time

Returns str :

time *secs* from now in Tanium SOAP API format

pytan.utils.**test_app_port**(*host*, *port*)
Validates that *host*:*port* can be reached using `port_check()`

Parameters **host** : str

hostname/ip address to check *port* on

port : int

port to check on *host*

Raises **HandlerError** : `pytan.utils.HandlerError`

if *host*:*port* can not be reached

pytan.utils.**version_check**(*reqver*)
Allows scripts using `pytan` to validate the version of the script aginst the version of `pytan`

Parameters **reqver** : str

string containing version number to check against `Exception`

Raises **Exception** : `Exception`

if `pytan.__version__` is not greater or equal to *reqver*

pytan.utils.**xml_pretty**(*x*)
Uses `xmltodict` to pretty print an XML str *x*

Parameters **x** : str

XML string to pretty print

Returns str :

The pretty printed string of *x*

pytan.utils.**xml_pretty_resultobj**(*x*)
Uses `xmltodict` to pretty print an the result-object element in XML str *x*

Parameters **x** : str

XML string to pretty print

Returns str :

The pretty printed string of result-object in *x*

pytan.utils.**xml_pretty_resultxml**(*x*)
Uses `xmltodict` to pretty print an the ResultXML element in XML str *x*

Parameters **x** : str

XML string to pretty print

Returns str :

The pretty printed string of ResultXML in *x*

Utility Functions: Argument Parsers for Command Line Scripts

pytan.utils.**setup_parser**(*desc*, *help=False*)
Method to setup the base `pytan.utils.CustomArgParse` class for command line scripts that use `pytan`. This establishes the basic arguments that are needed by all such scripts, such as:

- –help
- –username
- –password
- –host
- –port
- –loglevel
- –debugformat (not shown in –help)

pytan.utils.**setup_get_object_argparser**(*obj*, *doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to get objects.

pytan.utils.**setup_create_json_object_argparser**(*obj*, *doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to create objects from json files.

pytan.utils.**setup_delete_object_argparser**(*obj*, *doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to delete objects.

pytan.utils.**setup_ask_saved_argparser**(*doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to ask saved questions.

pytan.utils.**setup_stop_action_argparser**(*doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to stop actions.

pytan.utils.**setup_deploy_action_argparser**(*doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to deploy actions.

pytan.utils.**setup_get_result_argparser**(*doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to get results for questions or actions.

pytan.utils.**setup_ask_manual_argparser**(*doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser(), then add specific arguments for scripts that use pytan to ask manual questions.

pytan.utils.**add_ask_report_argparser**(*parser*)
Method to extend a pytan.utils.CustomArgParse class for command line scripts with arguments for scripts that need to supply export format subparsers for asking questions.

pytan.utils.**add_report_file_options**(*parser*)
Method to extend a pytan.utils.CustomArgParse class for command line scripts with arguments for scripts that need to supply export file and directory options.

pytan.utils.**add_get_object_report_argparser**(*parser*)
Method to extend a pytan.utils.CustomArgParse class for command line scripts with arguments for scripts that need to supply export format subparsers for getting objects.

pytan.utils.**get_grp_opts**(*parser*, *grp_names*)
Used to get arguments in *parser* that match argument group names in *grp_names*

Parameters **parser** : argparse.ArgParse

ArgParse object

grp_names : list of str

list of str of argument group names to get arguments for

Returns **grp_opts** : list of str

list of arguments gathered from argument group names in *grp_names*

pytan.utils.**process_create_json_object_args**(*parser*, *handler*, *obj*, *all_args*)
Process command line args supplied by user for create json object

Parameters **parser** : argparse.ArgParse

ArgParse object used to parse *all_args*

handler : pytan.handler.Handler

Instance of Handler created from command line args

obj : str

Object type for create json object

all_args : dict

dict of args parsed from *parser*

Returns **response** : taniumpy.object_types.base.BaseType

response from pytan.handler.Handler.create_from_json()

pytan.utils.**process_delete_object_args**(*parser*, *handler*, *obj*, *all_args*)
Process command line args supplied by user for delete object

Parameters **parser** : argparse.ArgParse

ArgParse object used to parse *all_args*

handler : pytan.handler.Handler

Instance of Handler created from command line args

obj : str

Object type for delete object

all_args : dict

dict of args parsed from *parser*

Returns **response** : taniumpy.object_types.base.BaseType

response from pytan.handler.Handler.delete()

pytan.utils.**process_get_object_args**(*parser*, *handler*, *obj*, *all_args*)
Process command line args supplied by user for get object

Parameters **parser** : argparse.ArgParse

ArgParse object used to parse *all_args*

handler : pytan.handler.Handler

Instance of Handler created from command line args

obj : str

Object type for get object

all_args : dict

dict of args parsed from *parser*

Returns **response** : taniumpy.object_types.base.BaseType

response from pytan.handler.Handler.get()

Utility Functions: Dehumanize human strings

pytan.utils.**dehumanize_package**(*package*)

Turns a package str into a package definition

Parameters **package** : str

A str that describes a package and optionally a selector and/or parameters

Returns **package_def** : dict

dict parsed from *sensors*

pytan.utils.**dehumanize_question_filters**(*question_filters*)

Turns a question_filters str or list of str into a question filter definition

Parameters **question_filters** : str, list of str

A str or list of str that describes a sensor for a question filter(s) and optionally a selector and/or filter

Returns **question_filter_defs** : list of dict

list of dict parsed from *question_filters*

pytan.utils.**dehumanize_question_options**(*question_options*)

Turns a question_options str or list of str into a question option definition

Parameters **question_options** : str, list of str

A str or list of str that describes question options

Returns **question_option_defs** : list of dict

list of dict parsed from *question_options*

pytan.utils.**dehumanize_sensors**(*sensors*, *key='sensors'*, *empty_ok=False*)

Turns a sensors str or list of str into a sensor definition

Parameters **sensors** : str, list of str

A str or list of str that describes a sensor(s) and optionally a selector, parameters, filter, and/or options

key : str, optional

Name of key that user should have provided *sensors* as

empty_ok : bool, optional

False: *sensors* is not allowed to be empty, throw `HumanParserError` if it is empty
True: *sensors* is allowed to be empty

Returns **sensor_defs** : list of dict

list of dict parsed from *sensors*

pytan.utils.**extract_filter**(*s*)
Extracts a filter from str *s*

Parameters **s** : str

A str that may or may not have a filter identified by ', that HUMAN VALUE'

Returns **s** : str

str *s* without the parsed_filter included

parsed_filter : dict

filter attributes mapped from filter from *s* if any found

pytan.utils.**extract_options**(*s*)
Extracts options from str *s*

Parameters **s** : str

A str that may or may not have options identified by ', opt:name[:value]'

Returns **s** : str

str *s* without the parsed_options included

parsed_options : list

options extracted from *s* if any found

pytan.utils.**extract_params**(*s*)
Extracts parameters from str *s*

Parameters **s** : str

A str that may or may not have parameters identified by {key=value}

Returns **s** : str

str *s* without the parsed_params included

parsed_params : list

parameters extracted from *s* if any found

pytan.utils.**extract_selector**(*s*)
Extracts a selector from str *s*

Parameters **s** : str

A str that may or may not have a selector in the beginning in the form of id:, name:, or :hash – if no selector found, name will be assumed as the default selector

Returns **s** : str

str *s* without the parsed_selector included

parsed_selector : str

selector extracted from *s*, or 'name' if none found

pytan.utils.**map_filter**(*filter_str*)
Maps a filter str against `constants.FILTER_MAPS`

Parameters **filter_str** : str

filter_str str that should be validated

Returns **filter_attrs** : dict

dict containing mapped filter attributes for SOAP API

pytan.utils.**map_option**(*opt*, *dest*)
Maps an opt str against `constants.OPTION_MAPS`

Parameters **opt** : str

option str that should be validated

dest : list of str

list of valid destinations (i.e. *filter* or *group*)

Returns **opt_attrs** : dict

dict containing mapped option attributes for SOAP API

pytan.utils.**map_options**(*options*, *dest*)
Maps a list of options using `map_option()`

Parameters **options** : list of str

list of str that should be validated

dest : list of str

list of valid destinations (i.e. *filter* or *group*)

Returns **mapped_options** : dict

dict of all mapped_options

Utility Functions: kwargs getters

pytan.utils.**get_ask_kwargs**(***kwargs*)
Gets QuestionAsker args from kwargs and returns a dict with just those matching args

Parameters ****kwargs** : dict

kwargs to get keys from

Returns **ask_kwargs** : dict

args from kwargs that are found in `pytan.constants.ASK_KWARGS`

pytan.utils.**get_kwargs_int**(*key*, *default=None*, ***kwargs*)
Gets key from kwargs and validates it is an int

Parameters **key** : str

key to get from kwargs

default : int, optional

default value to use if key not found in kwargs

****kwargs** : dict

kwargs to get key from

Returns **val** : int

value from key, or default if supplied

pytan.utils.**get_req_kwargs**(***kwargs*)

Gets SOAP API request args from kwargs and returns a dict with just those matching args

Parameters ****kwargs** : dict

kwargs to get keys from

Returns **req_kwargs** : dict

args from kwargs that are found in pytan.constants.REQ_KWARGS

Utility Functions: Object mappers

pytan.utils.**get_obj_map**(*objtype*)

Gets an object map for *objtype*

Parameters **objtype** : str

object type to get object map from in pytan.constants.GET_OBJ_MAP

Returns **obj_map** : dict

matching object map for *objtype* from pytan.constants.GET_OBJ_MAP

pytan.utils.**get_q_obj_map**(*qtype*)

Gets an object map for *qtype*

Parameters **qtype** : str

question type to get object map from in pytan.constants.Q_OBJ_MAP

Returns **obj_map** : dict

matching object map for *qtype* from pytan.constants.Q_OBJ_MAP

Utility Functions: TaniumPy objects

pytan.utils.**apply_options_obj**(*options*, *obj*, *dest*)

Updates an object with options

Parameters **options** : dict

dict containing options definition

obj : taniumpy.object_types.base.BaseType

TaniumPy object to apply *options* to

dest : list of str

list of valid destinations (i.e. *filter* or *group*)

Returns **obj** : taniumpy.object_types.base.BaseType

TaniumPy object updated with attributes from *options*

pytan.utils.**build_group_obj**(*q_filter_defs*, *q_option_defs*)

Creates a Group object from q_filter_defs and q_option_defs

Parameters **q_filter_defs** : list of dict

List of dict that are question filter definitions

q_option_defs : dict

dict of question filter options

Returns **group_obj** : taniumpy.object_types.group.Group

Group object with list of taniumpy.object_types.filter.Filter built from *q_filter_defs* and *q_option_defs*

pytan.utils.**build_manual_q**(*selectlist_obj*, *group_obj*)

Creates a Question object from selectlist_obj and group_obj

Parameters **selectlist_obj** : taniumpy.object_types.select_list.SelectList

SelectList object to add to Question object

group_obj : taniumpy.object_types.group.Group

Group object to add to Question object

Returns **add_q_obj** : taniumpy.object_types.question.Question

Question object built from selectlist_obj and group_obj

pytan.utils.**build_metadatalist_obj**(*properties*, *nameprefix*)

Creates a MetadataList object from properties

Parameters **properties** : list of list of strs

list of lists, each list having two strs - str 1: property key, str2: property value

nameprefix : str

prefix to insert in front of property key when creating MetadataItem

Returns **metadatalist_obj** : taniumpy.object_types.metadata_list.MetadataList

MetadataList object with list of taniumpy.object_types.metadata_item.MetadataItem built from *properties*

pytan.utils.**build_param_obj**(*key*, *val*, *delim=''*)

Creates a Parameter object from key and value, surrounding key with delim

Parameters **key** : str

key to use for parameter

value : str

value to use for parameter

delim : str

str to surround key with when adding to parameter object

Returns **param_obj** : taniumpy.object_types.parameter.Parameter

Parameter object built from key and val

pytan.utils.**build_param_objlist**(*obj*, *user_params*, *delim=''*, *derive_def=False*, *empty_ok=False*)

Creates a ParameterList object from user_params

Parameters **obj** : taniumpy.object_types.base.BaseType

TaniumPy object to verify parameters against

user_params : dict

dict describing key and value of user supplied params

delim : str

str to surround key with when adding to parameter object

derive_def : bool, optional

- False: Do not derive default values, and throw a `HandlerError` if user did not supply a value for a given parameter
- True: Try to derive a default value for each parameter if user did not supply one

empty_ok : bool, optional

- False: If user did not supply a value for a given parameter, throw a `HandlerError`
- True: If user did not supply a value for a given parameter, do not add the parameter to the ParameterList object

Returns **param_objlist** : `taniumpy.object_types.parameter_list.ParameterList`

ParameterList object with list of `taniumpy.object_types.parameter.Parameter` built from user_params

pytan.utils.**build_selectlist_obj**(*sensor_defs*)

Creates a SelectList object from sensor_defs

Parameters **sensor_defs** : list of dict

List of dict that are sensor definitions

Returns **select_objlist** : `taniumpy.object_types.select_list.SelectList`

SelectList object with list of `taniumpy.object_types.select.Select` built from *sensor_defs*

pytan.utils.**derive_param_default**(*obj_param*)

Derive a parameter default

Parameters **obj_param** : dict

parameter dict from TaniumPy object

Returns **def_val** : str

default value derived from obj_param

pytan.utils.**empty_obj**(*taniumpy_object*)

Validate that a given TaniumPy object is not empty

Parameters **taniumpy_object** : `taniumpy.object_types.base.BaseType`

object to check if empty

Returns bool

True if *taniumpy_object* is considered empty, False otherwise

pytan.utils.**get_filter_obj**(*sensor_def*)

Creates a Filter object from sensor_def

Parameters **sensor_def** : dict

dict containing sensor definition

Returns **filter_obj** : `taniumpy.object_types.filter.Filter`

Filter object created from *sensor_def*

pytan.utils.**get_obj_params**(*obj*)
Get the parameters from a TaniumPy object and JSON load them

obj [taniumpy.object_types.base.BaseType] TaniumPy object to get parameters from

Returns **params** : dict

JSON loaded dict of parameters from *obj*

pytan.utils.**question_progress**(*asker*, *pct*)
Call back method for taniumpy.question_asker.QuestionAsker.run() to report progress while waiting for results from a question

Parameters **asker** : taniumpy.question_asker.QuestionAsker

QuestionAsker instance

pct : float

Percentage completion of question

Utility Functions: Definition objects

pytan.utils.**check_dictkey**(*d*, *key*, *valid_types*, *valid_list_types*)
Yet another method to check a dictionary for a key

Parameters **d** : dict

dictionary to check for key

key : str

key to check for in d

valid_types : list of str

list of str of valid types for key

valid_list_types : list of str

if key is a list, validate that all values of list are in valid_list_types

pytan.utils.**chk_def_key**(*def_dict*, *key*, *keytypes*, *keysubtypes=None*, *req=False*)
Checks that def_dict has key

Parameters **def_dict** : dict

Definition dictionary

key : str

key to check for in def_dict

keytypes : list of str

list of str of valid types for key

keysubtypes : list of str

if key is a dict or list, validate that all values of dict or list are in keysubtypes

req : bool

- False: key does not have to be in def_dict
- True: key must be in def_dict, throw DefinitionParserError if not

pytan.utils.**parse_defs**(*defname*, *deftypes*, *strconv=None*, *empty_ok=True*, *defs=None*, ***kwargs*)
Parses and validates defs into new_defs

Parameters **defname** : str

Name of definition

deftypes : list of str

list of valid types that defs can be

strconv : str

if supplied, and defs is a str, turn defs into a dict with key = strconv, value = defs

empty_ok : bool

- True: defs is allowed to be empty
- False: defs is not allowed to be empty

Returns **new_defs** : list of dict

parsed and validated defs

pytan.utils.**val_package_def**(*package_def*)
Validates package definitions

Ensures package definition has a selector, and if a package definition has a params key, that key is valid

Parameters **package_def** : dict

package definition

pytan.utils.**val_q_filter_defs**(*q_filter_defs*)
Validates question filter definitions

Ensures each question filter definition has a selector, and if a question filter definition has a filter key, that key is valid

Parameters **q_filter_defs** : list of dict

list of question filter definitions

pytan.utils.**val_sensor_defs**(*sensor_defs*)
Validates sensor definitions

Ensures each sensor definition has a selector, and if a sensor definition has a params, options, or filter key, that each key is valid

Parameters **sensor_defs** : list of dict

list of sensor definitions

1.5.4 pytan Unit Tests

This contains unit tests for pytan.

These unit tests do not require a connection to a Tanium server in order to run.

class test_pytan_unit.**TestDehumanizeExtractionUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_extract_filter_invalid()

test_extract_filter_nofilter()

test_extract_filter_valid()

test_extract_options_invalid_option()

test_extract_options_many()

test_extract_options_missing_value_max_data_age()

test_extract_options_missing_value_value_type()

test_extract_options_nooptions()

test_extract_options_single()

test_extract_params()

test_extract_params_missing_seperator()

test_extract_params_multiparams()

test_extract_params_noparams()

test_extract_selector()

test_extract_selector_use_name_if_noselector()

class test_pytan_unit.**TestDehumanizeQuestionFilterUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_empty_filterlist()

test_empty_filterstr()

test_invalid_filter1()

test_invalid_filter2()

test_invalid_filter3()

test_multi_filter_list()

test_single_filter_list()

test_single_filter_str()

class test_pytan_unit.**TestDehumanizeQuestionOptionUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_empty_optionlist()

test_empty_optionstr()

test_invalid_option1()

test_invalid_option2()

test_option_list_many()

test_option_list_multi()

test_option_list_single()

test_option_str()

class test_pytan_unit.**TestDehumanizeSensorUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_empty_args_dict()

test_empty_args_list()

test_empty_args_str()

test_multi_list_complex()

test_single_str()

test_single_str_complex1()

test_single_str_complex2()

test_single_str_with_filter()

test_valid_simple_list()

test_valid_simple_str_hash_selector()

test_valid_simple_str_id_selector()

test_valid_simple_str_name_selector()

class test_pytan_unit.**TestGenericUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_ask_kwargs()

test_empty_obj()

test_get_now()

test_get_obj_map()

test_get_q_obj_map()

test_invalid_port()

test_is_dict()

test_is_list()

test_is_not_dict()

test_is_not_list()

test_is_not_num()

test_is_not_str()

test_is_num()

test_is_str()

test_jsonify()

test_req_kwargs()

test_version_higher()

test_version_lower()

class test_pytan_unit.**TestManualBuildObjectUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

classmethod setUpClass()

test_build_group_obj()

test_build_manual_q()

test_build_selectlist_obj_invalid_filter()

test_build_selectlist_obj_missing_value()

test_build_selectlist_obj_noparamssensorobj_noparams()
builds a selectlist object using a sensor obj with no params

test_build_selectlist_obj_noparamssensorobj_withparams()
builds a selectlist object using a sensor obj with no params, but passing in params (which should be ignored)

test_build_selectlist_obj_withparamssensorobj_noparams()
builds a selectlist object using a sensor obj with 4 params but not supplying any values for any of the params

test_build_selectlist_obj_withparamssensorobj_withparams()
builds a selectlist object using a sensor obj with 4 params but supplying a value for only one param

class test_pytan_unit.**TestManualPackageDefValidateUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_invalid1()

test_invalid2()

test_valid1()

test_valid2()

class test_pytan_unit.**TestManualQuestionFilterDefParseUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_parse_emptydict()

test_parse_emptylist()

test_parse_emptystr()

test_parse_multi_filter()

test_parse_noargs()

test_parse_none()

test_parse_single_filter()

test_parse_str()

class test_pytan_unit.**TestManualQuestionFilterDefValidateUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_invalid1()

test_valid1()

test_valid2()

class test_pytan_unit.**TestManualQuestionOptionDefParseUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_parse_emptydict()

test_parse_emptylist()

test_parse_emptystr()

test_parse_list()

test_parse_noargs()

test_parse_none()

test_parse_options_dict()

test_parse_str()

class test_pytan_unit.**TestManualSensorDefParseUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_parse_complex()
list with many items is parsed into same list

test_parse_dict_hash()
dict with hash is parsed into list of same dict

test_parse_dict_id()
dict with id is parsed into list of same dict

test_parse_dict_name()
dict with name is parsed into list of same dict

test_parse_emptydict()
args=={} throws exception

test_parse_emptylist()
args==[] throws exception

test_parse_emptystr()
args==" throws exception

test_parse_noargs()
no args throws exception

test_parse_none()
args==None throws exception

test_parse_str1()
simple str is parsed into list of same str

class test_pytan_unit.**TestManualSensorDefValidateUtils**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_unit'

test_invalid1()

test_invalid2()

test_invalid3()

test_invalid4()

test_valid1()

test_valid2()

test_valid3()

test_valid4()

1.5.5 pytan Functional Tests

This contains functional tests for pytan.

These functional tests require a connection to a Tanium server in order to run. The connection info is pulled from the SERVER_INFO dictionary in test/API_INFO.py.

class test_pytan_func.**CreateObjFromJsonTests**(*methodName='runTest'*)

Bases: unittest.case.TestCase

__module__ = 'test_pytan_func'

classmethod setUpClass()

setup_test()

test_create_from_json_action()

test_create_from_json_client()

test_create_from_json_group()

test_create_from_json_package()

test_create_from_json_question()

test_create_from_json_saved_action()

test_create_from_json_saved_question()

test_create_from_json_sensor()

test_create_from_json_setting()

test_create_from_json_user()

test_create_from_json_userrole()

test_create_from_json_whitelisted_url()

class test_pytan_func.**CreateObjectTests**(*methodName='runTest'*)

Bases: unittest.case.TestCase

__module__ = 'test_pytan_func'

classmethod setUpClass()

setup_test()

test_create_group()

test_create_package()

test_create_sensor()

test_create_user()

test_create_whitelisted_url()

class test_pytan_func.**ExportObjTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_func'

classmethod setUpClass()

setup_test()

test_export_basetype()

test_export_resultset()

class test_pytan_func.**InvalidServerTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_func'

classmethod setUpClass()

test_invalid_connect_1_bad_username()

test_invalid_connect_2_bad_host_and_non_ssl_port()

test_invalid_connect_3_bad_password()

test_invalid_connect_4_bad_host_and_bad_port()

class test_pytan_func.**ValidServerTests**(*methodName='runTest'*)
Bases: unittest.case.TestCase

__module__ = 'test_pytan_func'

classmethod setUpClass()

setup_test()

test_deploy_action_missing_package()

test_deploy_action_missing_params()

test_deploy_action_no_run()

test_invalid_get_object_1_get_question_object_fail_by_name()

test_invalid_get_object_2_get_action_object_single_by_name()

test_invalid_question_1_ask_manual_human_question_param_missing_keysplit()

test_invalid_question_2_ask_manual_question_invalid_sensor()

test_invalid_question_3_ask_manual_question_filterhelp()

test_invalid_question_4_ask_manual_human_question_invalid_sensor()

test_invalid_question_5_ask_manual_human_question_invalid_filter()

test_invalid_question_6_ask_manual_question_optionhelp()

test_invalid_question_7_ask_manual_human_question_toomanyparams()

test_valid_deploy_action()

test_valid_deploy_action_no_results()

test_valid_get_object_10_get_leader_clients()

test_valid_get_object_11_get_all_saved_questions()

test_valid_get_object_12_get_user_single_by_id()

test_valid_get_object_13_get_saved_action_single_by_name()

test_valid_get_object_14_get_all_settings()

test_valid_get_object_15_get_sensor_multiple_selectors()

test_valid_get_object_16_get_setting_single_by_name()

test_valid_get_object_17_get_all_userroless()

test_valid_get_object_18_get_all_questions()

test_valid_get_object_19_get_all_groups()

test_valid_get_object_1_get_all_users()

test_valid_get_object_20_get_all_sensors()

test_valid_get_object_21_get_action_single_by_id()

test_valid_get_object_22_get_all_whitelisted_urls()

test_valid_get_object_23_get_saved_question_single_by_name()

test_valid_get_object_24_get_sensor_multiple()

test_valid_get_object_25_get_user_single_by_name()

test_valid_get_object_26_get_all_clients()

test_valid_get_object_27_get_group_single_by_name()

test_valid_get_object_28_get_all_packages()

test_valid_get_object_29_get_all_actions()

test_valid_get_object_2_get_question_single_by_id()

test_valid_get_object_30_get_userrole_single_by_id()

test_valid_get_object_3_get_sensor_single_by_hash()

test_valid_get_object_4_get_sensor_single_by_id()

test_valid_get_object_5_get_package_single_by_name()

test_valid_get_object_6_get_sensor_single_by_name()

test_valid_get_object_7_get_saved_question_multiple()

test_valid_get_object_8_get_whitelisted_url_single_by_id()

test_valid_get_object_9_get_all_saved_actions()

test_valid_question_10_ask_manual_human_question_filter()

test_valid_question_11_ask_manual_human_question_params_single()

test_valid_question_12_ask_manual_human_question_simple()

test_valid_question_13_ask_manual_human_question_param_sensor_noparams()

test_valid_question_14_ask_manual_human_question_params_multiple()

test_valid_question_15_ask_manual_human_question_complex()

test_valid_question_16_ask_manual_human_question_paramsandfilterandoptions()

test_valid_question_1_ask_manual_human_question_options()

test_valid_question_2_ask_manual_human_question_nonparamsensor_params()

test_valid_question_3_ask_manual_human_question_multiple()

test_valid_question_4_ask_manual_human_question_filterandoptions()

test_valid_question_5_ask_manual_question_sensor_complex()

test_valid_question_6_ask_saved_question_single_list()

test_valid_question_7_ask_saved_question_single_str()

test_valid_question_8_ask_manual_human_question_paramsandfilter()

test_valid_question_9_ask_manual_human_question_multiple_selector()

test_pytan_func.**spew**(*m*)

1.6 taniumpy package

1.6.1 taniumpy.session module

Session handler for Tanium API

exception taniumpy.session.**AuthorizationError**
Bases: exceptions.Exception

exception taniumpy.session.**BadResponseError**
Bases: exceptions.Exception

class taniumpy.session.**DynamicFormatter**
Bases: string.Formatter

get_value(*key*, *args*, *kwargs*)

exception taniumpy.session.**HttpError**
Bases: exceptions.Exception

class taniumpy.session.**Session**(*server*, *port=443*)
Bases: object

ADD_OBJECT = 'AddObject'

AUTH_RES = '/auth'

DELETE_OBJECT = 'DeleteObject'

FORMATTER(*format_string*, **args*, ***kwargs*)

GET_OBJECT = 'GetObject'

GET_RESULT_DATA = 'GetResultData'

GET_RESULT_INFO = 'GetResultInfo'

INFO_RES = '/info.json'

REQUEST_BODY = u'<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd=

SOAP_PORT = 444

SOAP_RES = '/soap'

UPDATE_OBJECT = 'UpdateObject'

add(*obj*, ***kwargs*)

authenticate(*username=None*, *password=None*)

delete(*obj*, ***kwargs*)

find(*object_type*, ***kwargs*)

getResultData(*obj*, ***kwargs*)

getResultInfo(*obj*, ***kwargs*)

get_server_info()

is_auth

save(*obj*, ***kwargs*)

server_version

session_id

taniumpy.session.**http_post**(*host*, *port*, *url*, *body=None*, *headers=None*, *timeout=5*)

taniumpy.session.**load_file**(*filename*)

1.6.2 taniumpy.question_asker module

class taniumpy.question_asker.**QuestionAsker**(*session*, *question*, *polling_interval=None*, *pct_complete_threshold=99*, *timeout=300*)

Bases: object

A class to aid in asking a Question.

The primary function of this class is to poll for result info for question, and fire off events:

ProgressChanged AnswersChanged AnswersComplete

POLLING_INTERVAL = 5

run(*callbacks={}*, ***kwargs*)

Poll for question data and issue callbacks.

Callbacks should be a dict with members: 'ProgressChanged' 'AnswersChanged' 'AnswersComplete'

Each should be a function that accepts a QuestionAsker and a percent complete.

Any callback can choose to get data from the session by calling asker.session.getResultData(asker.question)

Polling will be stopped only when one of the callbacks calls the stop() method or the answers are complete. Note that callbacks can call setPercentCompleteThreshold to change what done means on the fly

setPctCompleteThreshold(*val*)

stop()

exception taniumpy.question_asker.**QuestionTimeoutException**

Bases: exceptions.Exception

1.6.3 taniumpy.object_types package

taniumpy.object_types module

taniumpy.object_types.action module

class taniumpy.object_types.action.**Action**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.action_list module

class taniumpy.object_types.action_list.**ActionList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.action_list_info module

class taniumpy.object_types.action_list_info.**ActionListInfo**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.action_stop module

class taniumpy.object_types.action_stop.**ActionStop**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.action_stop_list module

class taniumpy.object_types.action_stop_list.**ActionStopList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.all_objects module

taniumpy.object_types.archived_question module

class taniumpy.object_types.archived_question.**ArchivedQuestion**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.archived_question_list module

class taniumpy.object_types.archived_question_list.**ArchivedQuestionList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.audit_data module

class taniumpy.object_types.audit_data.**AuditData**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.base module

class taniumpy.object_types.base.**BaseType**(*simple_properties*, *complex_properties*, *list_properties*)

Bases: object

append(*n*)
Allow adding to list.

Only supported on types that have a single property that is in list_properties

explode_json(*val*)

flatten_jsonable(*val*, *prefix*)

classmethod fromSOAPBody(*body*)
Parse body (text) and produce Python tanium objects.

This method assumes a single result_object, which may be a list or a single object.

classmethod fromSOAPElement(*el*)

static from_jsonable(*jsonable*)
Inverse of to_jsonable, with explode_json_string_values=False.

This can be used to import objects from serialized JSON. This JSON should come from BaseType.to_jsonable(explode_json_string_values=False, include+type=True)

Examples

```
>>> with open('question_list.json') as fd:
...     questions = json.loads(fd.read())
...     # is a list of serialized questions
...     question_objects = BaseType.from_jsonable(questions)
...     # will return a list of api.Question
```

toSOAPBody(*minimal=False*)

toSOAPElement(*minimal=False*)

to_flat_dict(*prefix=''*, *explode_json_string_values=False*)
Convert the object to a dict, flattening any lists or nested types

to_flat_dict_explode_json(*val*, *prefix=''*)
see if the value is json. If so, flatten it out into a dict

static to_json(*jsonable*, ***kwargs*)
Convert to a json string.

jsonable can be a single BaseType instance of a list of BaseType

to_jsonable(*explode_json_string_values=False*, *include_type=True*)

static write_csv(*fd*, *val*, *explode_json_string_values=False*, ***kwargs*)
Write 'val' to CSV. val can be a BaseType instance or a list of BaseType

This does a two-pass, calling to_flat_dict for each object, then finding the union of all headers, then writing out the value of each column for each object sorted by header name

explode_json_string_values attempts to see if any of the str values are parseable by json.loads, and if so treat each property as a column value

fd is a file-like object

exception taniumpy.object_types.base.**IncorrectTypeException**(*property*, *expected*, *actual*)

Bases: exceptions.Exception

Raised when a property is not of the expected type

taniumpy.object_types.cache_filter module

class taniumpy.object_types.cache_filter.**CacheFilter**

Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.cache_filter_list module

class taniumpy.object_types.cache_filter_list.**CacheFilterList**

Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.cache_info module

class taniumpy.object_types.cache_info.**CacheInfo**

Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.client_count module

class taniumpy.object_types.client_count.**ClientCount**

Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.client_status module

class taniumpy.object_types.client_status.**ClientStatus**

Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.column module

class taniumpy.object_types.column.**Column**

Bases: object

classmethod fromSOAPElement(*el*)

taniumpy.object_types.column_set module

class taniumpy.object_types.column_set.**ColumnSet**

Bases: object

classmethod fromSOAPElement(*el*)

taniumpy.object_types.computer_group module

class taniumpy.object_types.computer_group.**ComputerGroup**

Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.computer_group_list module

class taniumpy.object_types.computer_group_list.**ComputerGroupList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.computer_group_spec module

class taniumpy.object_types.computer_group_spec.**ComputerGroupSpec**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.computer_spec_list module

class taniumpy.object_types.computer_spec_list.**ComputerSpecList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.error_list module

class taniumpy.object_types.error_list.**ErrorList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.filter module

class taniumpy.object_types.filter.**Filter**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.filter_list module

class taniumpy.object_types.filter_list.**FilterList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.group module

class taniumpy.object_types.group.**Group**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.group_list module

class taniumpy.object_types.group_list.**GroupList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.metadata_item module

class taniumpy.object_types.metadata_item.**MetadataItem**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.metadata_list module

class taniumpy.object_types.metadata_list.**MetadataList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.object_list module

class taniumpy.object_types.object_list.**ObjectList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.object_list_types module

taniumpy.object_types.options module

class taniumpy.object_types.options.**Options**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_file module

class taniumpy.object_types.package_file.**PackageFile**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_file_list module

class taniumpy.object_types.package_file_list.**PackageFileList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_file_status module

class taniumpy.object_types.package_file_status.**PackageFileStatus**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_file_status_list module

class taniumpy.object_types.package_file_status_list.**PackageFileStatusList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_file_template module

class taniumpy.object_types.package_file_template.**PackageFileTemplate**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_file_template_list module

class taniumpy.object_types.package_file_template_list.**PackageFileTemplateList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_spec module

class taniumpy.object_types.package_spec.**PackageSpec**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.package_spec_list module

class taniumpy.object_types.package_spec_list.**PackageSpecList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parameter module

class taniumpy.object_types.parameter.**Parameter**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parameter_list module

class taniumpy.object_types.parameter_list.**ParameterList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parse_job module

class taniumpy.object_types.parse_job.**ParseJob**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parse_job_list module

class taniumpy.object_types.parse_job_list.**ParseJobList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parse_result module

class taniumpy.object_types.parse_result.**ParseResult**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parse_result_group module

class taniumpy.object_types.parse_result_group.**ParseResultGroup**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parse_result_group_list module

class taniumpy.object_types.parse_result_group_list.**ParseResultGroupList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.parse_result_list module

class taniumpy.object_types.parse_result_list.**ParseResultList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin module

class taniumpy.object_types.plugin.**Plugin**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_argument module

class taniumpy.object_types.plugin_argument.**PluginArgument**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_argument_list module

class taniumpy.object_types.plugin_argument_list.**PluginArgumentList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_command_list module

class taniumpy.object_types.plugin_command_list.**PluginCommandList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_list module

class taniumpy.object_types.plugin_list.**PluginList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_schedule module

class taniumpy.object_types.plugin_schedule.**PluginSchedule**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_schedule_list module

class taniumpy.object_types.plugin_schedule_list.**PluginScheduleList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_sql module

class taniumpy.object_types.plugin_sql.**PluginSql**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_sql_column module

class taniumpy.object_types.plugin_sql_column.**PluginSqlColumn**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.plugin_sql_result module

class taniumpy.object_types.plugin_sql_result.**PluginSqlResult**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.question module

class taniumpy.object_types.question.**Question**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.question_list module

class taniumpy.object_types.question_list.**QuestionList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.question_list_info module

class taniumpy.object_types.question_list_info.**QuestionListInfo**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.result_info module

class taniumpy.object_types.result_info.**ResultInfo**
Bases: object

Wrap the result of GetResultInfo

classmethod fromSOAPElement(*el*)
Deserialize a ResultInfo from a result_info SOAPElement

Assumes all properties are integer values (true today)

taniumpy.object_types.result_set module

class taniumpy.object_types.result_set.**ResultSet**
Bases: object

Wrap the result of GetResultData

classmethod fromSOAPElement(*el*)
Deserialize a ResultInfo from a result_info SOAPElement

Assumes all properties are integer values (true today)

static to_json(*jsonable*, ***kwargs*)
Convert to a json string.

jsonable must be a ResultSet instance

to_jsonable(***kwargs*)

static write_csv(*fd*, *val*, ***kwargs*)

taniumpy.object_types.row module

class taniumpy.object_types.row.**Row**(*columns*)
Bases: object

A row in a result set.

Values are stored in column order, also accessible by key using []

classmethod fromSOAPElement(*el*, *columns*)

taniumpy.object_types.saved_action module

class taniumpy.object_types.saved_action.**SavedAction**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.saved_action_approval module

class taniumpy.object_types.saved_action_approval.**SavedActionApproval**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.saved_action_list module

class taniumpy.object_types.saved_action_list.**SavedActionList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.saved_action_policy module

class taniumpy.object_types.saved_action_policy.**SavedActionPolicy**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.saved_action_row_id_list module

class taniumpy.object_types.saved_action_row_id_list.**SavedActionRowIdList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.saved_question module

class taniumpy.object_types.saved_question.**SavedQuestion**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.saved_question_list module

class taniumpy.object_types.saved_question_list.**SavedQuestionList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.select module

class taniumpy.object_types.select.**Select**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.select_list module

class taniumpy.object_types.select_list.**SelectList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor module

class taniumpy.object_types.sensor.**Sensor**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor_list module

class taniumpy.object_types.sensor_list.**SensorList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor_query module

class taniumpy.object_types.sensor_query.**SensorQuery**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor_query_list module

class taniumpy.object_types.sensor_query_list.**SensorQueryList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor_string_hints module

class taniumpy.object_types.sensor_string_hints.**SensorStringHints**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor_subcolumn module

class taniumpy.object_types.sensor_subcolumn.**SensorSubcolumn**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor_subcolumn_list module

class taniumpy.object_types.sensor_subcolumn_list.**SensorSubcolumnList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.sensor_types module

taniumpy.object_types.soap_error module

class taniumpy.object_types.soap_error.**SoapError**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.system_setting module

class taniumpy.object_types.system_setting.**SystemSetting**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.system_settings_list module

class taniumpy.object_types.system_settings_list.**SystemSettingsList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.system_status_aggregate module

class taniumpy.object_types.system_status_aggregate.**SystemStatusAggregate**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.system_status_list module

class taniumpy.object_types.system_status_list.**SystemStatusList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.upload_file module

class taniumpy.object_types.upload_file.**UploadFile**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.upload_file_list module

class taniumpy.object_types.upload_file_list.**UploadFileList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.upload_file_status module

class taniumpy.object_types.upload_file_status.**UploadFileStatus**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.user module

class taniumpy.object_types.user.**User**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.user_list module

class taniumpy.object_types.user_list.**UserList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.user_permissions module

class taniumpy.object_types.user_permissions.**UserPermissions**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.user_role module

class taniumpy.object_types.user_role.**UserRole**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.user_role_list module

class taniumpy.object_types.user_role_list.**UserRoleList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.version_aggregate module

class taniumpy.object_types.version_aggregate.**VersionAggregate**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.version_aggregate_list module

class taniumpy.object_types.version_aggregate_list.**VersionAggregateList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.white_listed_url module

class taniumpy.object_types.white_listed_url.**WhiteListedUrl**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.white_listed_url_list module

class taniumpy.object_types.white_listed_url_list.**WhiteListedUrlList**
Bases: taniumpy.object_types.base.BaseType

taniumpy.object_types.xml_error module

class taniumpy.object_types.xml_error.**XmlError**
Bases: taniumpy.object_types.base.BaseType

1.7 xmltodict module

Makes working with XML feel like you are working with JSON

xmltodict.**parse**(*xml_input*, *encoding=None*, *expat=<module 'xml.parsers.expat' from '/Library/Python/2.7/site-packages/_xmlplus/parsers/expat.pyc'>*, *process_namespaces=False*, *namespace_separator=':'*, ***kwargs*)

Parse the given XML input and convert it into a dictionary.

xml_input can either be a *string* or a file-like object.

If *xml_attribs* is *True*, element attributes are put in the dictionary among regular child elements, using @ as a prefix to avoid collisions. If set to *False*, they are just ignored.

Simple example:

```
>>> import xmltodict
>>> doc = xmltodict.parse("""
... <a prop="x">
...   <b>1</b>
...   <b>2</b>
... </a>
... """)
>>> doc['a']['@prop']
u'x'
>>> doc['a']['b']
[u'1', u'2']
```

If *item_depth* is *0*, the function returns a dictionary for the root element (default behavior). Otherwise, it calls *item_callback* every time an item at the specified depth is found and returns *None* in the end (streaming mode).

The callback function receives two parameters: the *path* from the document root to the item (name-attribs pairs), and the *item* (dict). If the callback's return value is false-ish, parsing will be stopped with the `ParsingInterrupted` exception.

Streaming example:

```
>>> def handle(path, item):
...     print 'path:%s item:%s' % (path, item)
...     return True
...
>>> xmltodict.parse("""
... <a prop="x">
...   <b>1</b>
...   <b>2</b>
... </a>""", item_depth=2, item_callback=handle)
path:[(u'a', {u'prop': u'x'}), (u'b', None)] item:1
path:[(u'a', {u'prop': u'x'}), (u'b', None)] item:2
```

The optional argument *postprocessor* is a function that takes *path*, *key* and *value* as positional arguments and returns a new *(key, value)* pair where both *key* and *value* may have changed. Usage example:

```
>>> def postprocessor(path, key, value):
...     try:
...         return key + ':int', int(value)
...     except (ValueError, TypeError):
...         return key, value
>>> xmltodict.parse('<a><b>1</b><b>2</b><b>x</b></a>',
```

```
...                     postprocessor=postprocessor)
OrderedDict([(u'a', OrderedDict([(u'b:int', [1, 2]), (u'b', u'x')]))])
```

You can pass an alternate version of *expat* (such as *defusedexpat*) by using the *expat* parameter. E.g:

```
>>> import defusedexpat
>>> xmltodict.parse('<a>hello</a>', expat=defusedexpat.pyexpat)
OrderedDict([(u'a', u'hello')])
```

xmltodict.**unparse**(*input_dict*, *output=None*, *encoding='utf-8'*, *full_document=True*, ***kwargs*)
Emit an XML document for the given *input_dict* (reverse of *parse*).

The resulting XML document is returned as a string, but if *output* (a file-like object) is specified, it is written there instead.

Dictionary keys prefixed with *attr_prefix* (default=''@') *are interpreted as XML node attributes, whereas keys equal to 'cdata_key* (default=''#text'') are treated as character data.

The *pretty* parameter (default='False') enables pretty-printing. In this mode, lines are terminated with *'n'* and indented with *'t'*, but this can be customized with the *newl* and *indent* parameters.

1.8 ddt module

ddt.**data**(**values*)
Method decorator to add to your test methods.

Should be added to methods of instances of `unittest.TestCase`.

ddt.**ddt**(*cls*)
Class decorator for subclasses of `unittest.TestCase`.

Apply this decorator to the test case class, and then decorate test methods with `@data`.

For each method decorated with `@data`, this will effectively create as many methods as data items are passed as parameters to `@data`.

The names of the test methods follow the pattern `original_test_name_{ordinal}_{data}`. `ordinal` is the position of the data argument, starting with 1.

For data we use a string representation of the data value converted into a valid python identifier. If `data.__name__` exists, we use that instead.

For each method decorated with `@file_data('test_data.json')`, the decorator will try to load the test_data.json file located relative to the python file containing the method that is decorated. It will, for each `test_name` key create as many methods in the list of values from the `data` key.

ddt.**file_data**(*value*)
Method decorator to add to your test methods.

Should be added to methods of instances of `unittest.TestCase`.

`value` should be a path relative to the directory of the file containing the decorated `unittest.TestCase`. The file should contain JSON encoded data, that can either be a list or a dict.

In case of a list, each value in the list will correspond to one test case, and the value will be concatenated to the test method name.

In case of a dict, keys will be used as suffixes to the name of the test case, and values will be fed as test data.

ddt.**is_hash_randomized**()

ddt.**mk_test_name**(*name*, *value*, *index=0*)
Generate a new name for a test case.

It will take the original test name and append an ordinal index and a string representation of the value, and convert the result into a valid python identifier by replacing extraneous characters with _.

If hash randomization is enabled (a feature available since 2.7.3/3.2.3 and enabled by default since 3.3) and a "non-trivial" value is passed this will omit the name argument by default. Set *PYTHONHASHSEED* to a fixed value before running tests in these cases to get the names back consistently or use the *__name__* attribute on data values.

A "trivial" value is a plain scalar, or a tuple or list consisting only of trivial values.

ddt.**unpack**(*func*)
Method decorator to add unpack feature.

1.9 threaded_http module

Simple HTTP server for testing purposes

class threaded_http.**Handler**(*request*, *client_address*, *server*)
Bases: BaseHTTPServer.BaseHTTPRequestHandler

__module__ = 'threaded_http'

do_GET()

log_message(*format*, **args*)

class threaded_http.**ThreadedHTTPServer**(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*)
Bases: SocketServer.ThreadingMixIn, BaseHTTPServer.HTTPServer

Handle requests in a separate thread.

__module__ = 'threaded_http'

threaded_http.**threaded_http**(*host='localhost'*, *port=4443*, *verbosity=2*)
establishes an HTTP server on host:port in a thread

CHAPTER

TWO

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

d

ddt, 56

p

pytan, 2
pytan.constants, 17
pytan.handler, 2
pytan.utils, 19

t

taniumpy, 41
taniumpy.object_types, 43
taniumpy.object_types.action, 43
taniumpy.object_types.action_list, 43
taniumpy.object_types.action_list_info, 43
taniumpy.object_types.action_stop, 43
taniumpy.object_types.action_stop_list, 43
taniumpy.object_types.all_objects, 43
taniumpy.object_types.archived_question, 43
taniumpy.object_types.archived_question_list, 43
taniumpy.object_types.audit_data, 43
taniumpy.object_types.base, 44
taniumpy.object_types.cache_filter, 45
taniumpy.object_types.cache_filter_list, 45
taniumpy.object_types.cache_info, 45
taniumpy.object_types.client_count, 45
taniumpy.object_types.client_status, 45
taniumpy.object_types.column, 45
taniumpy.object_types.column_set, 45
taniumpy.object_types.computer_group, 45
taniumpy.object_types.computer_group_list, 46
taniumpy.object_types.computer_group_spec, 46
taniumpy.object_types.computer_spec_list, 46
taniumpy.object_types.error_list, 46
taniumpy.object_types.filter, 46
taniumpy.object_types.filter_list, 46
taniumpy.object_types.group, 46
taniumpy.object_types.group_list, 46
taniumpy.object_types.metadata_item, 46
taniumpy.object_types.metadata_list, 47
taniumpy.object_types.object_list, 47
taniumpy.object_types.object_list_types, 47
taniumpy.object_types.options, 47
taniumpy.object_types.package_file, 47
taniumpy.object_types.package_file_list, 47
taniumpy.object_types.package_file_status, 47
taniumpy.object_types.package_file_status_list, 47
taniumpy.object_types.package_file_template, 47
taniumpy.object_types.package_file_template_list, 47
taniumpy.object_types.package_spec, 48
taniumpy.object_types.package_spec_list, 48
taniumpy.object_types.parameter, 48
taniumpy.object_types.parameter_list, 48
taniumpy.object_types.parse_job, 48
taniumpy.object_types.parse_job_list, 48
taniumpy.object_types.parse_result, 48
taniumpy.object_types.parse_result_group, 48
taniumpy.object_types.parse_result_group_list, 48
taniumpy.object_types.parse_result_list, 49
taniumpy.object_types.plugin, 49
taniumpy.object_types.plugin_argument, 49

taniumpy.object_types.plugin_argument_list, 49
taniumpy.object_types.plugin_command_list, 49
taniumpy.object_types.plugin_list, 49
taniumpy.object_types.plugin_schedule, 49
taniumpy.object_types.plugin_schedule_list, 49
taniumpy.object_types.plugin_sql, 49
taniumpy.object_types.plugin_sql_column, 50
taniumpy.object_types.plugin_sql_result, 50
taniumpy.object_types.question, 50
taniumpy.object_types.question_list, 50
taniumpy.object_types.question_list_info, 50
taniumpy.object_types.result_info, 50
taniumpy.object_types.result_set, 50
taniumpy.object_types.row, 51
taniumpy.object_types.saved_action, 51
taniumpy.object_types.saved_action_approval, 51
taniumpy.object_types.saved_action_list, 51
taniumpy.object_types.saved_action_policy, 51
taniumpy.object_types.saved_action_row_id_list, 51
taniumpy.object_types.saved_question, 51
taniumpy.object_types.saved_question_list, 51
taniumpy.object_types.select, 52
taniumpy.object_types.select_list, 52
taniumpy.object_types.sensor, 52
taniumpy.object_types.sensor_list, 52
taniumpy.object_types.sensor_query, 52
taniumpy.object_types.sensor_query_list, 52
taniumpy.object_types.sensor_string_hints, 52
taniumpy.object_types.sensor_subcolumn, 52
taniumpy.object_types.sensor_subcolumn_list, 52
taniumpy.object_types.sensor_types, 53
taniumpy.object_types.soap_error, 53
taniumpy.object_types.system_setting, 53
taniumpy.object_types.system_settings_list, 53
taniumpy.object_types.system_status_aggregate, 53
taniumpy.object_types.system_status_list, 53
taniumpy.object_types.upload_file, 53
taniumpy.object_types.upload_file_list, 53
taniumpy.object_types.upload_file_status, 53
taniumpy.object_types.user, 53
taniumpy.object_types.user_list, 54
taniumpy.object_types.user_permissions, 54
taniumpy.object_types.user_role, 54
taniumpy.object_types.user_role_list, 54
taniumpy.object_types.version_aggregate, 54
taniumpy.object_types.version_aggregate_list, 54
taniumpy.object_types.white_listed_url, 54
taniumpy.object_types.white_listed_url_list, 54
taniumpy.object_types.xml_error, 54
taniumpy.question_asker, 42
taniumpy.session, 41
test_pytan_func, 38
test_pytan_unit, 33
threaded_http, 57

X

xmltodict, 55

Symbols

__author__ (in module pytan), 2
__copyright__ (in module pytan), 2
__license__ (in module pytan), 2
__module__ (test_pytan_func.CreateObjFromJsonTests attribute), 38
__module__ (test_pytan_func.CreateObjectTests attribute), 38
__module__ (test_pytan_func.ExportObjTests attribute), 39
__module__ (test_pytan_func.InvalidServerTests attribute), 39
__module__ (test_pytan_func.ValidServerTests attribute), 39
__module__ (test_pytan_unit.TestDehumanizeExtractionUtils attribute), 33
__module__ (test_pytan_unit.TestDehumanizeQuestionFilterUtils attribute), 34
__module__ (test_pytan_unit.TestDehumanizeQuestionOptionUtils attribute), 34
__module__ (test_pytan_unit.TestDehumanizeSensorUtils attribute), 35
__module__ (test_pytan_unit.TestGenericUtils attribute), 35
__module__ (test_pytan_unit.TestManualBuildObjectUtils attribute), 36
__module__ (test_pytan_unit.TestManualPackageDefValidateUtils attribute), 36
__module__ (test_pytan_unit.TestManualQuestionFilterDefParseUtils attribute), 36
__module__ (test_pytan_unit.TestManualQuestionFilterDefValidateUtils attribute), 36
__module__ (test_pytan_unit.TestManualQuestionOptionDefParseUtils attribute), 37
__module__ (test_pytan_unit.TestManualSensorDefParseUtils attribute), 37
__module__ (test_pytan_unit.TestManualSensorDefValidateUtils attribute), 37
__module__ (threaded_http.Handler attribute), 57
__module__ (threaded_http.ThreadedHTTPServer attribute), 57
__version__ (in module pytan), 2
_export_class_BaseType() (pytan.handler.Handler method), 17
_export_class_ResultSet() (pytan.handler.Handler method), 17
_export_format_csv() (pytan.handler.Handler method), 17
_export_format_json() (pytan.handler.Handler method), 17
_export_format_xml() (pytan.handler.Handler method), 17
_find() (pytan.handler.Handler method), 17
_get_multi() (pytan.handler.Handler method), 17
_get_package_def() (pytan.handler.Handler method), 17
_get_sensor_defs() (pytan.handler.Handler method), 17
_get_single() (pytan.handler.Handler method), 17
_single_find() (pytan.handler.Handler method), 17

A

Action (class in taniumpy.object_types.action), 43
ACTION_RESULT_STATUS (in module pytan.constants), 17
ActionList (class in taniumpy.object_types.action_list), 43
ActionListInfo (class in taniumpy.object_types.action_list_info), 43
ActionStop (class in taniumpy.object_types.action_stop), 43
ActionStopList (class in taniumpy.object_types.action_stop_list), 43
add() (taniumpy.session.Session method), 42
add_ask_report_argparser() (in module pytan.utils), 24
add_get_object_report_argparser() (in module pytan.utils), 24
ADD_OBJECT (taniumpy.session.Session attribute), 41
add_report_file_options() (in module pytan.utils), 24
append() (taniumpy.object_types.base.BaseType method), 44
apply_options_obj() (in module pytan.utils), 29
ArchivedQuestion (class in taniumpy.object_types.archived_question), 43
ArchivedQuestionList (class in taniumpy.object_types.archived_question_list), 43

ask() (pytan.handler.Handler method), 3
ASK_KWARGS (in module pytan.constants), 17
ask_manual() (pytan.handler.Handler method), 4
ask_manual_human() (pytan.handler.Handler method), 5
ask_saved() (pytan.handler.Handler method), 4
AuditData (class in taniumpy.object_types.audit_data), 43
AUTH_RES (taniumpy.session.Session attribute), 41
authenticate() (taniumpy.session.Session method), 42
AuthorizationError, 41

B

BadResponseError, 41
BaseType (class in taniumpy.object_types.base), 44
build_group_obj() (in module pytan.utils), 29
build_manual_q() (in module pytan.utils), 30
build_metadatalist_obj() (in module pytan.utils), 30
build_param_obj() (in module pytan.utils), 30
build_param_objlist() (in module pytan.utils), 30
build_selectlist_obj() (in module pytan.utils), 31

C

CacheFilter (class in taniumpy.object_types.cache_filter), 45
CacheFilterList (class in taniumpy.object_types.cache_filter_list), 45
CacheInfo (class in taniumpy.object_types.cache_info), 45
change_console_format() (in module pytan.utils), 20
check_dictkey() (in module pytan.utils), 32
chk_def_key() (in module pytan.utils), 32
ClientCount (class in taniumpy.object_types.client_count), 45
ClientStatus (class in taniumpy.object_types.client_status), 45
Column (class in taniumpy.object_types.column), 45
ColumnSet (class in taniumpy.object_types.column_set), 45
ComputerGroup (class in taniumpy.object_types.computer_group), 45
ComputerGroupList (class in taniumpy.object_types.computer_group_list), 46
ComputerGroupSpec (class in taniumpy.object_types.computer_group_spec), 46
ComputerSpecList (class in taniumpy.object_types.computer_spec_list), 46
create_from_json() (pytan.handler.Handler method), 9
create_group() (pytan.handler.Handler method), 13
create_package() (pytan.handler.Handler method), 13
create_sensor() (pytan.handler.Handler method), 14
create_user() (pytan.handler.Handler method), 14
create_whitelisted_url() (pytan.handler.Handler method), 15
CreateObjectTests (class in test_pytan_func), 38
CreateObjFromJsonTests (class in test_pytan_func), 38
CustomArgFormat (class in pytan.utils), 20
CustomArgParse (class in pytan.utils), 20

D

data() (in module ddt), 56
ddt (module), 56
ddt() (in module ddt), 56
DEBUG_FORMAT (in module pytan.constants), 17
DefinitionParserError, 19
dehumanize_package() (in module pytan.utils), 26
dehumanize_question_filters() (in module pytan.utils), 26
dehumanize_question_options() (in module pytan.utils), 26
dehumanize_sensors() (in module pytan.utils), 26
delete() (pytan.handler.Handler method), 15
delete() (taniumpy.session.Session method), 42
DELETE_OBJECT (taniumpy.session.Session attribute), 41
deploy_action() (pytan.handler.Handler method), 6
deploy_action_asker() (pytan.handler.Handler method), 9
deploy_action_human() (pytan.handler.Handler method), 8
derive_param_default() (in module pytan.utils), 31
do_GET() (threaded_http.Handler method), 57
DynamicFormatter (class in taniumpy.session), 41

E

emit() (pytan.utils.SplitStreamHandler method), 20
empty_obj() (in module pytan.utils), 31
error() (pytan.utils.CustomArgParse method), 20
ErrorList (class in taniumpy.object_types.error_list), 46
explode_json() (taniumpy.object_types.base.BaseType method), 44
EXPORT_MAPS (in module pytan.constants), 17
export_obj() (pytan.handler.Handler method), 10
export_to_report_file() (pytan.handler.Handler method), 11
ExportObjTests (class in test_pytan_func), 39
extract_filter() (in module pytan.utils), 27
extract_options() (in module pytan.utils), 27
extract_params() (in module pytan.utils), 27
extract_selector() (in module pytan.utils), 27

F

file_data() (in module ddt), 56
Filter (class in taniumpy.object_types.filter), 46
FILTER_MAPS (in module pytan.constants), 17
FILTER_RE (in module pytan.constants), 18
FilterList (class in taniumpy.object_types.filter_list), 46
find() (taniumpy.session.Session method), 42

flatten_jsonable() (taniumpy.object_types.base.BaseType method), 44
FORMATTER() (taniumpy.session.Session method), 41
from_jsonable() (taniumpy.object_types.base.BaseType static method), 44
fromSOAPBody() (taniumpy.object_types.base.BaseType class method), 44
fromSOAPElement() (taniumpy.object_types.base.BaseType class method), 44
fromSOAPElement() (taniumpy.object_types.column.Column class method), 45
fromSOAPElement() (taniumpy.object_types.column_set.ColumnSet class method), 45
fromSOAPElement() (taniumpy.object_types.result_info.ResultInfo class method), 50
fromSOAPElement() (taniumpy.object_types.result_set.ResultSet class method), 50
fromSOAPElement() (taniumpy.object_types.row.Row class method), 51

G

get() (pytan.handler.Handler method), 15
get_all() (pytan.handler.Handler method), 16
get_ask_kwargs() (in module pytan.utils), 28
get_dict_list_items() (in module pytan.utils), 21
get_dict_list_len() (in module pytan.utils), 21
get_filter_obj() (in module pytan.utils), 31
get_grp_opts() (in module pytan.utils), 25
get_kwargs_int() (in module pytan.utils), 28
get_now() (in module pytan.utils), 21
GET_OBJ_MAP (in module pytan.constants), 18
get_obj_map() (in module pytan.utils), 29
get_obj_params() (in module pytan.utils), 31
GET_OBJECT (taniumpy.session.Session attribute), 41
get_q_obj_map() (in module pytan.utils), 29
get_req_kwargs() (in module pytan.utils), 29
GET_RESULT_DATA (taniumpy.session.Session attribute), 41
get_result_data() (pytan.handler.Handler method), 16
GET_RESULT_INFO (taniumpy.session.Session attribute), 41
get_result_info() (pytan.handler.Handler method), 16
get_server_info() (taniumpy.session.Session method), 42
get_value() (taniumpy.session.DynamicFormatter method), 41
getResultData() (taniumpy.session.Session method), 42
getResultInfo() (taniumpy.session.Session method), 42
Group (class in taniumpy.object_types.group), 46
GroupList (class in taniumpy.object_types.group_list), 46

H

Handler (class in pytan.handler), 2
Handler (class in threaded_http), 57
HandlerError, 19
http_post() (in module taniumpy.session), 42
HttpError, 41
human_time() (in module pytan.utils), 22
HumanParserError, 19

I

IncorrectTypeException, 45
INFO_FORMAT (in module pytan.constants), 18
INFO_RES (taniumpy.session.Session attribute), 41
InvalidServerTests (class in test_pytan_func), 39
is_auth (taniumpy.session.Session attribute), 42
is_dict() (in module pytan.utils), 21
is_hash_randomized() (in module ddt), 56
is_list() (in module pytan.utils), 21
is_num() (in module pytan.utils), 21
is_str() (in module pytan.utils), 21

J

jsonify() (in module pytan.utils), 22

L

load_file() (in module taniumpy.session), 42
load_taniumpy_from_json() (pytan.handler.Handler method), 10
LOG_LEVEL_MAPS (in module pytan.constants), 18
log_message() (threaded_http.Handler method), 57

M

map_filter() (in module pytan.utils), 27
map_option() (in module pytan.utils), 28
map_options() (in module pytan.utils), 28
MetadataItem (class in taniumpy.object_types.metadata_item), 46
MetadataList (class in taniumpy.object_types.metadata_list), 47
mk_test_name() (in module ddt), 57

O

ObjectList (class in taniumpy.object_types.object_list), 47
OPTION_MAPS (in module pytan.constants), 18
OPTION_RE (in module pytan.constants), 19
Options (class in taniumpy.object_types.options), 47

P

PackageFile (class in taniumpy.object_types.package_file), 47

PackageFileList (class in taniumpy.object_types.package_file_list), 47
PackageFileStatus (class in taniumpy.object_types.package_file_status), 47
PackageFileStatusList (class in taniumpy.object_types.package_file_status_list), 47
PackageFileTemplate (class in taniumpy.object_types.package_file_template), 47
PackageFileTemplateList (class in taniumpy.object_types.package_file_template_list), 47
PackageSpec (class in taniumpy.object_types.package_spec), 48
PackageSpecList (class in taniumpy.object_types.package_spec_list), 48
PARAM_DELIM (in module pytan.constants), 19
PARAM_KEY_SPLIT (in module pytan.constants), 19
PARAM_RE (in module pytan.constants), 19
PARAM_SPLIT_RE (in module pytan.constants), 19
Parameter (class in taniumpy.object_types.parameter), 48
ParameterList (class in taniumpy.object_types.parameter_list), 48
parse() (in module xmltodict), 55
parse_defs() (in module pytan.utils), 32
ParseJob (class in taniumpy.object_types.parse_job), 48
ParseJobList (class in taniumpy.object_types.parse_job_list), 48
ParseResult (class in taniumpy.object_types.parse_result), 48
ParseResultGroup (class in taniumpy.object_types.parse_result_group), 48
ParseResultGroupList (class in taniumpy.object_types.parse_result_group_list), 48
ParseResultList (class in taniumpy.object_types.parse_result_list), 49
Plugin (class in taniumpy.object_types.plugin), 49
PluginArgument (class in taniumpy.object_types.plugin_argument), 49
PluginArgumentList (class in taniumpy.object_types.plugin_argument_list), 49
PluginCommandList (class in taniumpy.object_types.plugin_command_list), 49
PluginList (class in taniumpy.object_types.plugin_list), 49
PluginSchedule (class in taniumpy.object_types.plugin_schedule), 49
PluginScheduleList (class in taniumpy.object_types.plugin_schedule_list), 49
PluginSql (class in taniumpy.object_types.plugin_sql), 49
PluginSqlColumn (class in taniumpy.object_types.plugin_sql_column), 50
PluginSqlResult (class in taniumpy.object_types.plugin_sql_result), 50
POLLING_INTERVAL (taniumpy.question_asker.QuestionAsker attribute), 42
port_check() (in module pytan.utils), 22
print_help() (pytan.utils.CustomArgParse method), 20
process_create_json_object_args() (in module pytan.utils), 25
process_delete_object_args() (in module pytan.utils), 25
process_get_object_args() (in module pytan.utils), 25
pytan (module), 2
pytan.constants (module), 17
pytan.handler (module), 2
pytan.utils (module), 19

Q

Q_OBJ_MAP (in module pytan.constants), 19
Question (class in taniumpy.object_types.question), 50
question_progress() (in module pytan.utils), 32
QuestionAsker (class in taniumpy.question_asker), 42
QuestionList (class in taniumpy.object_types.question_list), 50
QuestionListInfo (class in taniumpy.object_types.question_list_info), 50
QuestionTimeoutException, 42

R

remove_logging_handler() (in module pytan.utils), 20
REQ_KWARGS (in module pytan.constants), 19
REQUEST_BODY (taniumpy.session.Session attribute), 41
ResultInfo (class in taniumpy.object_types.result_info), 50
ResultSet (class in taniumpy.object_types.result_set), 50
Row (class in taniumpy.object_types.row), 51
run() (taniumpy.question_asker.QuestionAsker method), 42
RunFalse, 19

S

save() (taniumpy.session.Session method), 42
SavedAction (class in taniumpy.object_types.saved_action), 51
SavedActionApproval (class in taniumpy.object_types.saved_action_approval), 51

SavedActionList (class in taniumpy.object_types.saved_action_list), 51
SavedActionPolicy (class in taniumpy.object_types.saved_action_policy), 51
SavedActionRowIdList (class in taniumpy.object_types.saved_action_row_id_list), 51
SavedQuestion (class in taniumpy.object_types.saved_question), 51
SavedQuestionList (class in taniumpy.object_types.saved_question_list), 51
seconds_from_now() (in module pytan.utils), 22
Select (class in taniumpy.object_types.select), 52
SelectList (class in taniumpy.object_types.select_list), 52
SELECTORS (in module pytan.constants), 19
Sensor (class in taniumpy.object_types.sensor), 52
SENSOR_TYPE_MAP (in module pytan.constants), 19
SensorList (class in taniumpy.object_types.sensor_list), 52
SensorQuery (class in taniumpy.object_types.sensor_query), 52
SensorQueryList (class in taniumpy.object_types.sensor_query_list), 52
SensorStringHints (class in taniumpy.object_types.sensor_string_hints), 52
SensorSubcolumn (class in taniumpy.object_types.sensor_subcolumn), 52
SensorSubcolumnList (class in taniumpy.object_types.sensor_subcolumn_list), 52
server_version (taniumpy.session.Session attribute), 42
Session (class in taniumpy.session), 41
session_id (taniumpy.session.Session attribute), 42
set_all_loglevels() (in module pytan.utils), 20
set_log_levels() (in module pytan.utils), 20
setPctCompleteThreshold() (taniumpy.question_asker.QuestionAsker method), 42
setup_ask_manual_argparser() (in module pytan.utils), 24
setup_ask_saved_argparser() (in module pytan.utils), 24
setup_console_logging() (in module pytan.utils), 21
setup_create_json_object_argparser() (in module pytan.utils), 24
setup_delete_object_argparser() (in module pytan.utils), 24
setup_deploy_action_argparser() (in module pytan.utils), 24
setup_get_object_argparser() (in module pytan.utils), 24
setup_get_result_argparser() (in module pytan.utils), 24
setup_parser() (in module pytan.utils), 23
setup_stop_action_argparser() (in module pytan.utils), 24
setup_test() (test_pytan_func.CreateObjectTests method), 38
setup_test() (test_pytan_func.CreateObjFromJsonTests method), 38
setup_test() (test_pytan_func.ExportObjTests method), 39
setup_test() (test_pytan_func.ValidServerTests method), 39
setUpClass() (test_pytan_func.CreateObjectTests class method), 38
setUpClass() (test_pytan_func.CreateObjFromJsonTests class method), 38
setUpClass() (test_pytan_func.ExportObjTests class method), 39
setUpClass() (test_pytan_func.InvalidServerTests class method), 39
setUpClass() (test_pytan_func.ValidServerTests class method), 39
setUpClass() (test_pytan_unit.TestManualBuildObjectUtils class method), 36
SOAP_PORT (taniumpy.session.Session attribute), 41
SOAP_RES (taniumpy.session.Session attribute), 41
SoapError (class in taniumpy.object_types.soap_error), 53
spew() (in module test_pytan_func), 41
SplitStreamHandler (class in pytan.utils), 20
stop() (taniumpy.question_asker.QuestionAsker method), 42
stop_action() (pytan.handler.Handler method), 9
SystemSetting (class in taniumpy.object_types.system_setting), 53
SystemSettingsList (class in taniumpy.object_types.system_settings_list), 53
SystemStatusAggregate (class in taniumpy.object_types.system_status_aggregate), 53
SystemStatusList (class in taniumpy.object_types.system_status_list), 53

T

taniumpy (module), 41
taniumpy.object_types (module), 43
taniumpy.object_types.action (module), 43
taniumpy.object_types.action_list (module), 43
taniumpy.object_types.action_list_info (module), 43
taniumpy.object_types.action_stop (module), 43
taniumpy.object_types.action_stop_list (module), 43
taniumpy.object_types.all_objects (module), 43
taniumpy.object_types.archived_question (module), 43
taniumpy.object_types.archived_question_list (module), 43
taniumpy.object_types.audit_data (module), 43
taniumpy.object_types.base (module), 44

taniumpy.object_types.cache_filter (module), 45
taniumpy.object_types.cache_filter_list (module), 45
taniumpy.object_types.cache_info (module), 45
taniumpy.object_types.client_count (module), 45
taniumpy.object_types.client_status (module), 45
taniumpy.object_types.column (module), 45
taniumpy.object_types.column_set (module), 45
taniumpy.object_types.computer_group (module), 45
taniumpy.object_types.computer_group_list (module), 46
taniumpy.object_types.computer_group_spec (module), 46
taniumpy.object_types.computer_spec_list (module), 46
taniumpy.object_types.error_list (module), 46
taniumpy.object_types.filter (module), 46
taniumpy.object_types.filter_list (module), 46
taniumpy.object_types.group (module), 46
taniumpy.object_types.group_list (module), 46
taniumpy.object_types.metadata_item (module), 46
taniumpy.object_types.metadata_list (module), 47
taniumpy.object_types.object_list (module), 47
taniumpy.object_types.object_list_types (module), 47
taniumpy.object_types.options (module), 47
taniumpy.object_types.package_file (module), 47
taniumpy.object_types.package_file_list (module), 47
taniumpy.object_types.package_file_status (module), 47
taniumpy.object_types.package_file_status_list (module), 47
taniumpy.object_types.package_file_template (module), 47
taniumpy.object_types.package_file_template_list (module), 47
taniumpy.object_types.package_spec (module), 48
taniumpy.object_types.package_spec_list (module), 48
taniumpy.object_types.parameter (module), 48
taniumpy.object_types.parameter_list (module), 48
taniumpy.object_types.parse_job (module), 48
taniumpy.object_types.parse_job_list (module), 48
taniumpy.object_types.parse_result (module), 48
taniumpy.object_types.parse_result_group (module), 48
taniumpy.object_types.parse_result_group_list (module), 48
taniumpy.object_types.parse_result_list (module), 49
taniumpy.object_types.plugin (module), 49
taniumpy.object_types.plugin_argument (module), 49
taniumpy.object_types.plugin_argument_list (module), 49
taniumpy.object_types.plugin_command_list (module), 49
taniumpy.object_types.plugin_list (module), 49
taniumpy.object_types.plugin_schedule (module), 49
taniumpy.object_types.plugin_schedule_list (module), 49
taniumpy.object_types.plugin_sql (module), 49
taniumpy.object_types.plugin_sql_column (module), 50
taniumpy.object_types.plugin_sql_result (module), 50
taniumpy.object_types.question (module), 50
taniumpy.object_types.question_list (module), 50
taniumpy.object_types.question_list_info (module), 50
taniumpy.object_types.result_info (module), 50
taniumpy.object_types.result_set (module), 50
taniumpy.object_types.row (module), 51
taniumpy.object_types.saved_action (module), 51
taniumpy.object_types.saved_action_approval (module), 51
taniumpy.object_types.saved_action_list (module), 51
taniumpy.object_types.saved_action_policy (module), 51
taniumpy.object_types.saved_action_row_id_list (module), 51
taniumpy.object_types.saved_question (module), 51
taniumpy.object_types.saved_question_list (module), 51
taniumpy.object_types.select (module), 52
taniumpy.object_types.select_list (module), 52
taniumpy.object_types.sensor (module), 52
taniumpy.object_types.sensor_list (module), 52
taniumpy.object_types.sensor_query (module), 52
taniumpy.object_types.sensor_query_list (module), 52
taniumpy.object_types.sensor_string_hints (module), 52
taniumpy.object_types.sensor_subcolumn (module), 52
taniumpy.object_types.sensor_subcolumn_list (module), 52
taniumpy.object_types.sensor_types (module), 53
taniumpy.object_types.soap_error (module), 53
taniumpy.object_types.system_setting (module), 53
taniumpy.object_types.system_settings_list (module), 53
taniumpy.object_types.system_status_aggregate (module), 53
taniumpy.object_types.system_status_list (module), 53
taniumpy.object_types.upload_file (module), 53
taniumpy.object_types.upload_file_list (module), 53
taniumpy.object_types.upload_file_status (module), 53
taniumpy.object_types.user (module), 53
taniumpy.object_types.user_list (module), 54
taniumpy.object_types.user_permissions (module), 54
taniumpy.object_types.user_role (module), 54
taniumpy.object_types.user_role_list (module), 54
taniumpy.object_types.version_aggregate (module), 54
taniumpy.object_types.version_aggregate_list (module), 54
taniumpy.object_types.white_listed_url (module), 54
taniumpy.object_types.white_listed_url_list (module), 54
taniumpy.object_types.xml_error (module), 54
taniumpy.question_asker (module), 42
taniumpy.session (module), 41
test_app_port() (in module pytan.utils), 23
test_ask_kwargs() (test_pytan_unit.TestGenericUtils method), 35
test_build_group_obj() (test_pytan_unit.TestManualBuildObjectUtils method), 36

test_build_manual_q() (test_pytan_unit.TestManualBuildObjectUtils method), 36
test_build_selectlist_obj_invalid_filter() (test_pytan_unit.TestManualBuildObjectUtils method), 36
test_build_selectlist_obj_missing_value() (test_pytan_unit.TestManualBuildObjectUtils method), 36
test_build_selectlist_obj_noparamssensorobj_noparams() (test_pytan_unit.TestManualBuildObjectUtils method), 36
test_build_selectlist_obj_noparamssensorobj_withparams() (test_pytan_unit.TestManualBuildObjectUtils method), 36
test_build_selectlist_obj_withparamssensorobj_noparams() (test_pytan_unit.TestManualBuildObjectUtils method), 36
test_build_selectlist_obj_withparamssensorobj_withparams() (test_pytan_unit.TestManualBuildObjectUtils method), 36
test_create_from_json_action() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_client() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_group() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_package() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_question() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_saved_action() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_saved_question() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_sensor() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_setting() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_user() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_userrole() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_from_json_whitelisted_url() (test_pytan_func.CreateObjFromJsonTests method), 38
test_create_group() (test_pytan_func.CreateObjectTests method), 38
test_create_package() (test_pytan_func.CreateObjectTests method), 38
test_create_sensor() (test_pytan_func.CreateObjectTests method), 38
test_create_user() (test_pytan_func.CreateObjectTests method), 39
test_create_whitelisted_url() (test_pytan_func.CreateObjectTests method), 39
test_deploy_action_missing_package() (test_pytan_func.ValidServerTests method), 39
test_deploy_action_missing_params() (test_pytan_func.ValidServerTests method), 39
test_deploy_action_no_run() (test_pytan_func.ValidServerTests method), 39
test_empty_args_dict() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_empty_args_list() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_empty_args_str() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_empty_filterlist() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_empty_filterstr() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_empty_obj() (test_pytan_unit.TestGenericUtils method), 35
test_empty_optionlist() (test_pytan_unit.TestDehumanizeQuestionOptionUt method), 34
test_empty_optionstr() (test_pytan_unit.TestDehumanizeQuestionOptionUt method), 34
test_export_basetype() (test_pytan_func.ExportObjTests method), 39
test_export_resultset() (test_pytan_func.ExportObjTests method), 39
test_extract_filter_invalid() (test_pytan_unit.TestDehumanizeExtractionUtils method), 33
test_extract_filter_nofilter() (test_pytan_unit.TestDehumanizeExtractionUtils method), 33
test_extract_filter_valid() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_options_invalid_option() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_options_many() (test_pytan_unit.TestDehumanizeExtractionUtils

method), 34
test_extract_options_missing_value_max_data_age() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_options_missing_value_value_type() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_options_nooptions() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_options_single() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_params() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_params_missing_seperator() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_params_multiparams() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_params_noparams() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_selector() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_extract_selector_use_name_if_noselector() (test_pytan_unit.TestDehumanizeExtractionUtils method), 34
test_get_now() (test_pytan_unit.TestGenericUtils method), 35
test_get_obj_map() (test_pytan_unit.TestGenericUtils method), 35
test_get_q_obj_map() (test_pytan_unit.TestGenericUtils method), 35
test_invalid1() (test_pytan_unit.TestManualPackageDefValidateUtils method), 36
test_invalid1() (test_pytan_unit.TestManualQuestionFilterDefValidateUtils method), 36
test_invalid1() (test_pytan_unit.TestManualSensorDefValidateUtils method), 37
test_invalid2() (test_pytan_unit.TestManualPackageDefValidateUtils method), 36
test_invalid2() (test_pytan_unit.TestManualSensorDefValidateUtils method), 38
test_invalid3() (test_pytan_unit.TestManualSensorDefValidateUtils method), 38
test_invalid4() (test_pytan_unit.TestManualSensorDefValidateUtils method), 38
test_invalid_connect_1_bad_username() (test_pytan_func.InvalidServerTests method), 39
test_invalid_connect_2_bad_host_and_non_ssl_port() (test_pytan_func.InvalidServerTests method), 39
test_invalid_connect_3_bad_password() (test_pytan_func.InvalidServerTests method), 39
test_invalid_connect_4_bad_host_and_bad_port() (test_pytan_func.InvalidServerTests method), 39
test_invalid_filter1() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_invalid_filter2() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_invalid_filter3() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_invalid_get_object_1_get_question_object_fail_by_name() (test_pytan_func.ValidServerTests method), 39
test_invalid_get_object_2_get_action_object_single_by_name() (test_pytan_func.ValidServerTests method), 39
test_invalid_option1() (test_pytan_unit.TestDehumanizeQuestionOptionUtil method), 34
test_invalid_option2() (test_pytan_unit.TestDehumanizeQuestionOptionUtil method), 34
test_invalid_port() (test_pytan_unit.TestGenericUtils method), 35
test_invalid_question_1_ask_manual_human_question_param_missing_key (test_pytan_func.ValidServerTests method), 39
test_invalid_question_2_ask_manual_question_invalid_sensor() (test_pytan_func.ValidServerTests method), 39
test_invalid_question_3_ask_manual_question_filterhelp() (test_pytan_func.ValidServerTests method), 39
test_invalid_question_4_ask_manual_human_question_invalid_sensor() (test_pytan_func.ValidServerTests method), 39
test_invalid_question_5_ask_manual_human_question_invalid_filter() (test_pytan_func.ValidServerTests method), 39
test_invalid_question_6_ask_manual_question_optionhelp() (test_pytan_func.ValidServerTests method), 39
test_invalid_question_7_ask_manual_human_question_toomanyparams() (test_pytan_func.ValidServerTests method), 39
test_is_dict() (test_pytan_unit.TestGenericUtils method), 35
test_is_list() (test_pytan_unit.TestGenericUtils method), 35
test_is_not_dict() (test_pytan_unit.TestGenericUtils method), 35
test_is_not_list() (test_pytan_unit.TestGenericUtils method), 35
test_is_not_num() (test_pytan_unit.TestGenericUtils method), 35
test_is_not_str() (test_pytan_unit.TestGenericUtils method), 35
test_is_num() (test_pytan_unit.TestGenericUtils method), 35
test_is_str() (test_pytan_unit.TestGenericUtils method), 35

test_jsonify() (test_pytan_unit.TestGenericUtils method), 35
test_multi_filter_list() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_multi_list_complex() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_option_list_many() (test_pytan_unit.TestDehumanizeQuestionOptionUtils method), 34
test_option_list_multi() (test_pytan_unit.TestDehumanizeQuestionOptionUtils method), 34
test_option_list_single() (test_pytan_unit.TestDehumanizeQuestionOptionUtils method), 34
test_option_str() (test_pytan_unit.TestDehumanizeQuestionOptionUtils method), 34
test_parse_complex() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_dict_hash() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_dict_id() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_dict_name() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_emptydict() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_emptydict() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_emptydict() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_emptylist() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_emptylist() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_emptylist() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_emptystr() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_emptystr() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_emptystr() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_list() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_multi_filter() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_noargs() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_noargs() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_noargs() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_none() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_none() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_none() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_parse_options_dict() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_single_filter() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_str() (test_pytan_unit.TestManualQuestionFilterDefParseUtils method), 36
test_parse_str() (test_pytan_unit.TestManualQuestionOptionDefParseUtils method), 37
test_parse_str1() (test_pytan_unit.TestManualSensorDefParseUtils method), 37
test_pytan_func (module), 38
test_pytan_unit (module), 33
test_reqkwargs() (test_pytan_unit.TestGenericUtils method), 35
test_single_filter_list() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_single_filter_str() (test_pytan_unit.TestDehumanizeQuestionFilterUtils method), 34
test_single_str() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_single_str_complex1() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_single_str_complex2() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_single_str_filter() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_valid1() (test_pytan_unit.TestManualPackageDefValidateUtils method), 36
test_valid1() (test_pytan_unit.TestManualQuestionFilterDefValidateUtils method), 37
test_valid1() (test_pytan_unit.TestManualSensorDefValidateUtils method), 38
test_valid2() (test_pytan_unit.TestManualPackageDefValidateUtils method), 36
test_valid2() (test_pytan_unit.TestManualQuestionFilterDefValidateUtils method), 37
test_valid2() (test_pytan_unit.TestManualSensorDefValidateUtils method), 38
test_valid3() (test_pytan_unit.TestManualSensorDefValidateUtils method), 38
test_valid4() (test_pytan_unit.TestManualSensorDefValidateUtils method), 38
test_valid_deploy_action() (test_pytan_func.ValidServerTests method), 39
test_valid_deploy_action_no_results() (test_pytan_func.ValidServerTests method), 39

test_valid_get_object_10_get_leader_clients() (test_pytan_func.ValidServerTests method), 39
test_valid_get_object_11_get_all_saved_questions() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_12_get_user_single_by_id() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_13_get_saved_action_single_by_name() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_14_get_all_settings() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_15_get_sensor_multiple_selectors() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_16_get_setting_single_by_name() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_17_get_all_userroless() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_18_get_all_questions() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_19_get_all_groups() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_1_get_all_users() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_20_get_all_sensors() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_21_get_action_single_by_id() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_22_get_all_whitelisted_urls() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_23_get_saved_question_single_by_name() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_24_get_sensor_multiple() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_25_get_user_single_by_name() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_26_get_all_clients() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_27_get_group_single_by_name() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_28_get_all_packages() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_29_get_all_actions() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_2_get_question_single_by_id() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_30_get_userrole_single_by_id() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_3_get_sensor_single_by_hash() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_4_get_sensor_single_by_id() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_5_get_package_single_by_name() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_6_get_sensor_single_by_name() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_7_get_saved_question_multiple() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_8_get_whitelisted_url_single_by_id() (test_pytan_func.ValidServerTests method), 40
test_valid_get_object_9_get_all_saved_actions() (test_pytan_func.ValidServerTests method), 40
test_valid_question_10_ask_manual_human_question_filter() (test_pytan_func.ValidServerTests method), 40
test_valid_question_11_ask_manual_human_question_params_single() (test_pytan_func.ValidServerTests method), 40
test_valid_question_12_ask_manual_human_question_simple() (test_pytan_func.ValidServerTests method), 40
test_valid_question_13_ask_manual_human_question_param_sensor_nopa (test_pytan_func.ValidServerTests method), 40
test_valid_question_14_ask_manual_human_question_params_multiple() (test_pytan_func.ValidServerTests method), 40
test_valid_question_15_ask_manual_human_question_complex() (test_pytan_func.ValidServerTests method), 40
test_valid_question_16_ask_manual_human_question_paramsandfilterando (test_pytan_func.ValidServerTests method), 40
test_valid_question_1_ask_manual_human_question_options() (test_pytan_func.ValidServerTests method), 41
test_valid_question_2_ask_manual_human_question_nonparamsensor_para (test_pytan_func.ValidServerTests method), 41
test_valid_question_3_ask_manual_human_question_multiple() (test_pytan_func.ValidServerTests method), 41
test_valid_question_4_ask_manual_human_question_filterandoptions() (test_pytan_func.ValidServerTests method), 41
test_valid_question_5_ask_manual_question_sensor_complex() (test_pytan_func.ValidServerTests method), 41
test_valid_question_6_ask_saved_question_single_list() (test_pytan_func.ValidServerTests method), 41
test_valid_question_7_ask_saved_question_single_str() (test_pytan_func.ValidServerTests method), 41
test_valid_question_8_ask_manual_human_question_paramsandfilter() (test_pytan_func.ValidServerTests method), 41
test_valid_question_9_ask_manual_human_question_multiple_selector() (test_pytan_func.ValidServerTests method), 41
test_valid_simple_list() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_valid_simple_str_hash_selector() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_valid_simple_str_id_selector() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_valid_simple_str_name_selector() (test_pytan_unit.TestDehumanizeSensorUtils method), 35
test_version_higher() (test_pytan_unit.TestGenericUtils method), 35
test_version_lower() (test_pytan_unit.TestGenericUtils method), 35

TestDehumanizeExtractionUtils (class in test_pytan_unit), 33
TestDehumanizeQuestionFilterUtils (class in test_pytan_unit), 34
TestDehumanizeQuestionOptionUtils (class in test_pytan_unit), 34
TestDehumanizeSensorUtils (class in test_pytan_unit), 34
TestGenericUtils (class in test_pytan_unit), 35
TestManualBuildObjectUtils (class in test_pytan_unit), 35
TestManualPackageDefValidateUtils (class in test_pytan_unit), 36
TestManualQuestionFilterDefParseUtils (class in test_pytan_unit), 36
TestManualQuestionFilterDefValidateUtils (class in test_pytan_unit), 36
TestManualQuestionOptionDefParseUtils (class in test_pytan_unit), 37
TestManualSensorDefParseUtils (class in test_pytan_unit), 37
TestManualSensorDefValidateUtils (class in test_pytan_unit), 37
threaded_http (module), 57
threaded_http() (in module threaded_http), 57
ThreadedHTTPServer (class in threaded_http), 57
to_flat_dict() (taniumpy.object_types.base.BaseType method), 44
to_flat_dict_explode_json() (taniumpy.object_types.base.BaseType method), 44
to_json() (taniumpy.object_types.base.BaseType static method), 44
to_json() (taniumpy.object_types.result_set.ResultSet static method), 50
to_jsonable() (taniumpy.object_types.base.BaseType method), 44
to_jsonable() (taniumpy.object_types.result_set.ResultSet method), 50
toSOAPBody() (taniumpy.object_types.base.BaseType method), 44
toSOAPElement() (taniumpy.object_types.base.BaseType method), 44

U

unpack() (in module ddt), 57
unparse() (in module xmltodict), 56
UPDATE_OBJECT (taniumpy.session.Session attribute), 42
UploadFile (class in taniumpy.object_types.upload_file), 53
UploadFileList (class in taniumpy.object_types.upload_file_list), 53
UploadFileStatus (class in taniumpy.object_types.upload_file_status), 53
User (class in taniumpy.object_types.user), 53
UserList (class in taniumpy.object_types.user_list), 54
UserPermissions (class in taniumpy.object_types.user_permissions), 54
UserRole (class in taniumpy.object_types.user_role), 54
UserRoleList (class in taniumpy.object_types.user_role_list), 54

V

val_package_def() (in module pytan.utils), 33
val_q_filter_defs() (in module pytan.utils), 33
val_sensor_defs() (in module pytan.utils), 33
ValidServerTests (class in test_pytan_func), 39
version_check() (in module pytan.utils), 23
VersionAggregate (class in taniumpy.object_types.version_aggregate), 54
VersionAggregateList (class in taniumpy.object_types.version_aggregate_list), 54

W

WhiteListedUrl (class in taniumpy.object_types.white_listed_url), 54
WhiteListedUrlList (class in taniumpy.object_types.white_listed_url_list), 54
write_csv() (taniumpy.object_types.base.BaseType static method), 44
write_csv() (taniumpy.object_types.result_set.ResultSet static method), 51

X

xml_pretty() (in module pytan.utils), 23
xml_pretty_resultobj() (in module pytan.utils), 23
xml_pretty_resultxml() (in module pytan.utils), 23
XmlError (class in taniumpy.object_types.xml_error), 54
xmltodict (module), 55