PyTan Documentation

Release 2.1.0

Jim Olsen

September 04, 2015

CHAPTER

ONE

TABLE OF CONTENTS

1.1 PyTan Introduction

1.1.1 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. For Tanium version 6.2: The SOAP servers listens on port 444 but is also available via port 443. For Tanium version 6.5: The SOAP servers listens on port 443, and is not available on port 444
- *TaniumPy Python Package*: (`taniumpy`) A python package comprised of a set of python objects automatically generated from the WSDL file that describes 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.

1.1.2 Why it was created

This was created to solve for the following needs:

- Create a python package (`pytan`) 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 `pytan` package 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.

1.1.3 Python Versions Supported

Python v3.x is not supported. PyTan has been fully tested against the following Python versions on OS X, Linux, and Windows:

- 2.7.6

- 2.7.9
- 2.7.10

1.1.4 Tanium Platform Versions Supported

PyTan has been fully tested against the following versions of the Tanium Platform:

- 6.2.314.3315
- 6.2.314.3321
- 6.5.314.4254
- 6.5.314.4268
- 6.5.314.4275

1.1.5 Installation

Windows Installation

- Download a supported Python version from https://www.python.org/downloads/windows/
- Install Python – 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 2.7.6 out of the box
- Copy PyTan from github to your local machine somewhere

Linux Installation

- Ensure Python 2.7.x is installed (most distributions ship with some variant of Python)
- Ensure the first python binary in your path points to your Python 2.7.x installation
- Copy PyTan from github to your local machine somewhere

1.1.6 Usage

- For command line usage, refer to Command Line Help Index
- For API Examples, refer to the PyTan API Examples
- For in depth API Documentation, refer to the PyTan Package, especially the *pytan.handler*

1.1.7 Directory Layout

- *EXAMPLES/ directory*: contains a set of example python files that show how to use the various methods exposed by (`pytan`)
- *BUILD/ directory*: contains the scripts that build the HTML and PDF documentation in doc/, generate the (`taniumpy`), generate the python examples in EXAMPLES/, generate some of the command line scripts in bin/, and generate all of the documentation for the command line scripts in doc/_static/bin_doc
- *bin/ directory*: contains all of the command line scripts that utilize the (`pytan`)
- *doc/ directory*: contains the HTML and PDF documentation
- *lib/ directory*: contains the python libraries (`pytan`) and (`taniumpy`), as well as other python libraries
- *test/ directory*: contains the unit and functional tests for (`pytan`)
- *winbin/ directory*: contains the Windows batch scripts which wrap around the python command line scripts in bin/
- *ZIP_DIST/ directory*: contains standalone windows executables for certain tools, created by batch files in BUILD/STATICWINBUILD/

1.1.8 Other References

- Tanium Platform Website
- Tanium Knowledge Base
- Tanium SOAP Knowledge Base Article
- The console.wsdl used to build the taniumpy library for this version, also useful as a reference tool.

1.2 PyTan Package

A python package that makes using the Tanium Server SOAP API easy.

`pytan.__version__` = **'2.1.0'**
Version of PyTan

`pytan.__copyright__` = **'Copyright 2015 Tanium'**
Copyright for PyTan

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

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

1.2.1 pytan.handler

The main `pytan` module that provides first level entities for programmatic use.

class `pytan.handler.`**`Handler`**(*username=None*, *password=None*, *host=None*, *port=443*, *loglevel=0*, *debugformat=False*, *gmt_log=True*, *session_id=None*, ***kwargs*)
Bases: `object`

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

Parametersusername : str

- default: None
- *username* to connect to *host* with

password : str

- default: None
- *password* to connect to *host* with

host : str

- default: None
- hostname or ip of Tanium SOAP Server

port : int, optional

- default: 443
- port of Tanium SOAP Server on *host*

loglevel : int, optional

- default: 0
- 0 do not print anything except warnings/errors
- 1 and higher will print more

debugformat : bool, optional

- default: False
- False: use one line logformat
- True: use two lines

gmt_log : bool, optional

- default: True
- True: use GMT timezone for log output
- False: use local time for log output

session_id : str, optional

- default: None
- session_id to use while authenticating instead of username/password

Other Parametershttp_debug : bool, optional

- default: False
- False: do not print requests package debug
- True: do print requests package debug
- This is passed through to *pytan.sessions.Session*

http_auth_retry: bool, optional

- default: True
- True: retry HTTP GET/POST's
- False: do not retry HTTP GET/POST's

- This is passed through to *pytan.sessions.Session*

http_retry_count: int, optional

- default: 5
- number of times to retry HTTP GET/POST's if the connection times out/fails
- This is passed through to *pytan.sessions.Session*

soap_request_headers : dict, optional

- default: {'Content-Type': 'text/xml; charset=utf-8', 'Accept-Encoding': 'gzip'}
- dictionary of headers to add to every HTTP GET/POST
- This is passed through to *pytan.sessions.Session*

auth_connect_timeout_sec : int, optional

- default: 5
- number of seconds before timing out for a connection while authenticating
- This is passed through to *pytan.sessions.Session*

auth_response_timeout_sec : int, optional

- default: 15
- number of seconds before timing out for a response while authenticating
- This is passed through to *pytan.sessions.Session*

info_connect_timeout_sec : int, optional

- default: 5
- number of seconds before timing out for a connection while getting /info.json
- This is passed through to *pytan.sessions.Session*

info_response_timeout_sec : int, optional

- default: 15
- number of seconds before timing out for a response while getting /info.json
- This is passed through to *pytan.sessions.Session*

soap_connect_timeout_sec : int, optional

- default: 15
- number of seconds before timing out for a connection for a SOAP request
- This is passed through to *pytan.sessions.Session*

soap_response_timeout_sec : int, optional

- default: 540
- number of seconds before timing out for a response for a SOAP request
- This is passed through to *pytan.sessions.Session*

stats_loop_enabled : bool, optional

- default: False
- False: do not enable the statistics loop thread

- True: enable the statistics loop thread
- This is passed through to *pytan.sessions.Session*

stats_loop_sleep_sec : int, optional

- default: 5
- number of seconds to sleep in between printing the statistics when stats_loop_enabled is True
- This is passed through to *pytan.sessions.Session*

record_all_requests: bool, optional

- default: False
- False: do not add each requests response object to session.ALL_REQUESTS_RESPONSES
- True: add each requests response object to session.ALL_REQUESTS_RESPONSES
- This is passed through to *pytan.sessions.Session*

stats_loop_targets : list of dict, optional

- default: [{'Version': 'Settings/Version'}, {'Active Questions': 'Active Question Cache/Active Question Estimate'}, {'Clients': 'Active Question Cache/Active Client Estimate'}, {'Strings': 'String Cache/Total String Count'}, {'Handles': 'System Performance Info/HandleCount'}, {'Processes': 'System Performance Info/ProcessCount'}, {'Memory Available': 'percentage(System Performance Info/PhysicalAvailable,System Performance Info/PhysicalTotal)'}]
- list of dictionaries with the key being the section of info.json to print info from, and the value being the item with in that section to print the value
- This is passed through to *pytan.sessions.Session*

persistent: bool, optional

- default: False
- False: do not request a persistent session
- True: do request a persistent
- This is passed through to *pytan.sessions.Session.authenticate()*

See also:

pytan.constants.LOG_LEVEL_MAPSmaps a given *loglevel* to respective logger names and their logger levels

pytan.constants.INFO_FORMATdebugformat=False

pytan.constants.DEBUG_FORMATdebugformat=True

taniumpy.session.Session Session object used by Handler

Notes

- for 6.2: 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. However, port 444 is the only port that exposes the /info page in 6.2
- for 6.5: port 443 is the default SOAP port, there is no port 444

Examples

Setup a Handler() object:

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

_add(*obj*, ***kwargs*)

Wrapper for interfacing with `taniumpy.session.Session.add()`

Parametersobj : *taniumpy.object_types.base.BaseType*

- object to add

Returnsadded_obj : *taniumpy.object_types.base.BaseType*

- full object that was added

_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()* instead.

Parameterssensor_defs : str, dict, list of str or dict

- default: []
- sensor definitions

question_filter_defs : dict, list of dict, optional

- default: []
- question filter definitions

question_option_defs : dict, list of dict, optional

- default: []
- question option definitions

get_results : bool, optional

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

sse : bool, optional

- default: False
- True: perform a server side export when getting result data
- False: perform a normal get result data (default for 6.2)
- Keeping False by default for now until the columnset's are properly identified in the server export

sse_format : str, optional

- default: 'xml_obj'
- format to have server side export report in, one of: {'csv', 'xml', 'xml_obj', 'cef', 0, 1, 2}

leading : str, optional

- default: ‘’
- used for sse_format ‘cef’ only, the string to prepend to each row

trailing : str, optional

- default: ‘’
- used for sse_format ‘cef’ only, the string to append to each row

polling_secs : int, optional

- default: 5
- Number of seconds to wait in between GetResultInfo loops
- This is passed through to *pytan.pollers.QuestionPoller*

complete_pct : int/float, optional

- default: 99
- Percentage of mr_tested out of estimated_total to consider the question “done”
- This is passed through to *pytan.pollers.QuestionPoller*

override_timeout_secs : int, optional

- default: 0
- If supplied and not 0, timeout in seconds instead of when object expires
- This is passed through to *pytan.pollers.QuestionPoller*

callbacks : dict, optional

- default: {}
- can be a dict of functions to be run with the key names being the various state changes: ‘ProgressChanged’, ‘AnswersChanged’, ‘AnswersComplete’
- This is passed through to *pytan.pollers.QuestionPoller.run()*

Returnsret : dict, containing:

- *question_object* : *taniumpy.object_types.question.Question*, the actual question created and added by PyTan
- *question_results* : *taniumpy.object_types.result_set.ResultSet*, the Result Set for *question_object* if *get_results* == True
- *poller_object* : *pytan.pollers.QuestionPoller*, poller object used to wait until all results are in before getting *question_results*
- *poller_success* : None if *get_results* == True, elsewise True or False

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 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': '.*'
... }
```

_check_sse_crash_prevention(*obj*, ***kwargs*)

Runs a number of methods used to prevent crashing the platform server when performing server side exports

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to pass to self._check_sse_empty_rs

_check_sse_empty_rs(*obj*, *ok_version*, ***kwargs*)

Checks if the server version is less than any versions in pytan.constants.SSE_CRASH_MAP, if so verifies that the result set is not empty

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to get result info for to ensure non-empty answers

ok_version : bool

•if the version currently running is an "ok" version

_check_sse_format_support(*sse_format*, *sse_format_int*, ***kwargs*)

Determines if the export format integer is supported in the server version

Parametersse_format : str or int

•user supplied export format

sse_format_int : int

•*sse_format* parsed into an int

_check_sse_timing(*ok_version*, ***kwargs*)

Checks that the last server side export was at least 1 second ago if server version is less than any versions in pytan.constants.SSE_CRASH_MAP

Parametersok_version : bool

•if the version currently running is an "ok" version

_check_sse_version(***kwargs*)
Validates that the server version supports server side export

_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()` instead.

Parameterspackage_def : dict

•definition that describes a package

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

•default: []

•action filter definitions

action_option_defs : dict, list of dict, optional

•default: []

•action filter option definitions

start_seconds_from_now : int, optional

•default: 0

•start action N seconds from now

expire_seconds : int, optional

•default: package.expire_seconds

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

run : bool, optional

•default: False

•False: just ask the question that pertains to verify action, export the results to CSV, and raise pytan.exceptions.RunFalse – does not deploy the action

•True: actually deploy the action

get_results : bool, optional

•default: True

•True: wait for result completion after deploying action

•False: just deploy the action and return the object in *ret*

action_name : str, optional

•default: prepend package name with "API Deploy "

•custom name for action

action_comment : str, optional

•default:

•custom comment for action

Returnsret : dict, containing:

•*saved_action_object* : `taniumpy.object_types.saved_action.SavedAction`, the saved_action added for this action (None if 6.2)

- *action_object* : *taniumpy.object_types.action.Action*, the action object that tanium created for *saved_action*
- *package_object* : taniumpy.object_types.package_spec.PackageSPec, the package object used in *saved_action*
- *action_info* : *taniumpy.object_types.result_info.ResultInfo*, the initial GetResultInfo call done before getting results
- *poller_object* : *pytan.pollers.ActionPoller*, poller object used to wait until all results are in before getting *action_results*
- *poller_success* : None if *get_results* == False, elsewise True or False
- *action_results* : None if *get_results* == False, elsewise *taniumpy.object_types.result_set.ResultSet*, the results for *action_object*
- *action_result_map* : None if *get_results* == False, elsewise progress map for *action_object* in dictionary form

See also:

pytan.constants.FILTER_MAPS valid filter dictionaries for filters

pytan.constants.OPTION_MAPS valid option dictionaries for options

Notes

- **For 6.2:**
  - We need to add an Action object
  - The Action object should not be in an ActionList
  - Action.start_time must be specified, if it is not specified the action shows up as expired immediately. We default to 1 second from current time if start_seconds_from_now is not passed in
- **For 6.5 / 6.6:**
  - We need to add a SavedAction object, the server creates the Action object for us
  - To emulate what the console does, the SavedAction should be in a SavedActionList
  - Action.start_time does not need to be specified

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}
        ... }
```

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

Parametersobj : *taniumpy.object_types.base.BaseType*

•taniumpy object to export

export_format : str

•str of format to perform export in

Returnsresult : str

•results of exporting *obj* into format *export_format*

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

Parametersobj : *taniumpy.object_types.result_set.ResultSet*

•taniumpy object to export

export_format : str

•str of format to perform export in

Returnsresult : str

•results of exporting *obj* into format *export_format*

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

Parametersobj : *taniumpy.object_types.result_set.ResultSet* or *taniumpy.object_types.base.BaseType*

•taniumpy object to export

Returnsresult : str

•results of exporting *obj* into csv format

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

Parametersobj : *taniumpy.object_types.result_set.ResultSet* or *taniumpy.object_types.base.BaseType*

•taniumpy object to export

Returnsresult : str

•results of exporting *obj* into json format

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

Parametersobj : *taniumpy.object_types.result_set.ResultSet* or *taniumpy.object_types.base.BaseType*

•taniumpy object to export

Returnsresult : str

•results of exporting *obj* into XML format

_find(*obj*, ***kwargs*)
Wrapper for interfacing with `taniumpy.session.Session.find()`

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to find

Returnsfound : *taniumpy.object_types.base.BaseType*

•full object that was found

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

Parametersobj_map : dict

•dict containing the map for a given object type

Returnsfound : *taniumpy.object_types.base.BaseType*

•full object that was found

_get_package_def(*d*, ***kwargs*)
Uses *get()* to update a definition with a package object

Parametersd : dict

•dict containing package definition

Returnsd : dict

•dict containing package definitions with package object in 'package_obj'

_get_sensor_defs(*defs*, ***kwargs*)
Uses *get()* to update a definition with a sensor object

Parametersdefs : list of dict

•list of dicts containing sensor definitions

Returnsdefs : list of dict

•list of dicts containing sensor definitions with sensor object in 'sensor_obj'

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

Parametersobj_map : dict

•dict containing the map for a given object type

Returnsfound : *taniumpy.object_types.base.BaseType*

•full object that was found

_resolve_sse_format(*sse_format*, ***kwargs*)
Resolves the server side export format the user supplied to an integer for the API

Parameterssse_format : str or int

•user supplied export format

Returnssse_format_int : int

•*sse_format* parsed into an int

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

Parametersobj_map : dict

•dict containing the map for a given object type

k : str

•attribute name to set to *v*

v : str

•attribute value to set on *k*

Returnsfound : *taniumpy.object_types.base.BaseType*

•full object that was found

_version_support_check(*v_maps*, ***kwargs*)
Checks that each of the version maps in v_maps is greater than or equal to the current servers version

Parametersv_maps : list of str

•each str should be a platform version

•each str will be checked against self.session.server_version

•if any str is not greater than or equal to self.session.server_version, return will be False

•if all strs are greater than or equal to self.session.server_version, return will be True

•if self.server_version is invalid/can't be determined, return will be False

Returnsbool

•True if all values in all v_maps are greater than or equal to self.session.server_version

•False otherwise

approve_saved_action(*id*, ***kwargs*)
Approve a saved action

Parametersid : int

•id of saved action to approve

Returnssaved_action_approve_obj : *taniumpy.object_types.saved_action_approval.SavedActio*

•The object containing the return from SavedActionApproval

ask(***kwargs*)
Ask a type of question and get the results back

Parametersqtype : str, optional

•default: 'manual'

•type of question to ask: {'saved', 'manual', '_manual'}

Returnsresult : 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()

pytan.handler.Handler.ask_saved() method used when qtype == 'saved'

pytan.handler.Handler.ask_manual() method used when qtype == 'manual'

pytan.handler.Handler._ask_manual() method used when qtype == '_manual'

ask_manual (***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()*

Parameterssensors : str, list of str

- default: []
- sensors (columns) to include in question

question_filters : str, list of str, optional

- default: []
- filters that apply to the whole question

question_options : str, list of str, optional

- default: []
- options that apply to the whole question

get_results : bool, optional

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

sensors_help : bool, optional

- default: False
- False: do not print the help string for sensors
- True: print the help string for sensors and exit

filters_help : bool, optional

- default: False
- False: do not print the help string for filters
- True: print the help string for filters and exit

options_help : bool, optional

- default: False
- False: do not print the help string for options
- True: print the help string for options and exit

polling_secs : int, optional

- default: 5
- Number of seconds to wait in between GetResultInfo loops

•This is passed through to *pytan.pollers.QuestionPoller*

complete_pct : int/float, optional

•default: 99

•Percentage of mr_tested out of estimated_total to consider the question "done"

•This is passed through to *pytan.pollers.QuestionPoller*

override_timeout_secs : int, optional

•default: 0

•If supplied and not 0, timeout in seconds instead of when object expires

•This is passed through to *pytan.pollers.QuestionPoller*

callbacks : dict, optional

•default: {}

•can be a dict of functions to be run with the key names being the various state changes: 'ProgressChanged', 'AnswersChanged', 'AnswersComplete'

•This is passed through to *pytan.pollers.QuestionPoller.run()*

Returnsresult : dict, containing:

•*question_object* : *taniumpy.object_types.question.Question*, the actual question created and added by PyTan

•*question_results* : *taniumpy.object_types.result_set.ResultSet*, the Result Set for *question_object* if *get_results* == True

•*poller_object* : *pytan.pollers.QuestionPoller*, poller object used to wait until all results are in before getting *question_results*

•*poller_success* : None if *get_results* == True, elsewise True or False

See also:

pytan.constants.FILTER_MAPS valid filter dictionaries for filters

pytan.constants.OPTION_MAPS valid option dictionaries for options

pytan.handler.Handler._ask_manual() private method with the actual workflow used to create and add the question object

Notes

When asking a question from the Tanium console, you construct a question like:

Get Computer Name and IP Route Details from all machines with Is Windows containing "True"

Asking the same question in PyTan has some similarities:

```
>>> r = handler.ask_manual(sensors=['Computer Name', 'IP Route Details'], question_filters=[
```

There are two sensors in this question, after the "Get" and before the "from all machines": "Computer Name" and "IP Route Details". The sensors after the "Get" and before the "from all machines" can be referred to as any number of things:

•sensors

•left hand side

•column selects

The sensors that are defined after the "Get" and before the "from all machines" are best described as a column selection, and control what columns you want to show up in your results. These sensor names are the same ones that would need to be passed into ask_question() for the sensors arguments.

You can filter your column selections by using a filter in the console like so:

> Get Computer Name starting with "finance" and IP Route Details from all machines with Is Windows containing "True"

And in PyTan:

```
>>> r = handler.ask_manual(sensors=['Computer Name, that starts with:finance', 'IP Route Det
```

This will cause the results to have the same number of columns, but for any machine that returns results that do not match the filter specified for a given sensor, the row for that column will contain "[no results]".

There is also a sensor specified after the "from all machines with": "Is Windows". This sensor can be referred to as any number of things:

•question filters

•sensors (also)

•right hand side

•row selects

Any system that does not match the conditions in the question filters will return no results at all. These question filters are really just sensors all over again, but instead of controlling what columns are output in the results, they control what rows are output in the results.

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']
```

ask_parsed(*question_text*, *picker=None*, *get_results=True*, ***kwargs*)

Ask a parsed question as *question_text* and use the index of the parsed results from *picker*

Parametersquestion_text : str

•The question text you want the server to parse into a list of parsed results

picker : int

•default: None

•The index number of the parsed results that correlates to the actual question you wish to run

get_results : bool, optional

•default: True

•True: wait for result completion after asking question

•False: just ask the question and return it in *ret*

sse : bool, optional

•default: False

•True: perform a server side export when getting result data

•False: perform a normal get result data (default for 6.2)

•Keeping False by default for now until the columnset's are properly identified in the server export

sse_format : str, optional

•default: 'xml_obj'

•format to have server side export report in, one of: {'csv', 'xml', 'xml_obj', 'cef', 0, 1, 2}

leading : str, optional

•default: ''

•used for sse_format 'cef' only, the string to prepend to each row

trailing : str, optional

•default: ''

•used for sse_format 'cef' only, the string to append to each row

polling_secs : int, optional

•default: 5

•Number of seconds to wait in between GetResultInfo loops

•This is passed through to *pytan.pollers.QuestionPoller*

complete_pct : int/float, optional

•default: 99

•Percentage of mr_tested out of estimated_total to consider the question "done"

•This is passed through to *pytan.pollers.QuestionPoller*

override_timeout_secs : int, optional

•default: 0

•If supplied and not 0, timeout in seconds instead of when object expires

•This is passed through to *pytan.pollers.QuestionPoller*

callbacks : dict, optional

•default: {}

•can be a dict of functions to be run with the key names being the various state changes: 'ProgressChanged', 'AnswersChanged', 'AnswersComplete'

•This is passed through to *pytan.pollers.QuestionPoller.run()*

Returnsret : dict, containing:

•*question_object* : *taniumpy.object_types.question.Question*, the actual question added by PyTan

•*question_results* : *taniumpy.object_types.result_set.ResultSet*, the Result Set for *question_object* if *get_results* == True

•*poller_object* : *pytan.pollers.QuestionPoller*, poller object used to wait until all results are in before getting *question_results*

•*poller_success* : None if *get_results* == True, elsewise True or False

Examples

Ask the server to parse 'computer name', but don't pick a choice (will print out a list of choices at critical logging lev

```
>>> v = handler.ask_parsed('computer name')
```

Ask the server to parse 'computer name' and pick index 1 as the question you want to run:

```
>>> v = handler.ask_parsed('computer name', picker=1)
```

ask_saved(*refresh_data=False*, ***kwargs*)

Ask a saved question and get the results back

Parametersid : int, list of int, optional

•id of saved question to ask

name : str, list of str

•name of saved question

refresh_data: bool, optional

•default False

•False: do not perform a getResultInfo before issuing a getResultData

•True: perform a getResultInfo before issuing a getResultData

sse : bool, optional

•default: False

•True: perform a server side export when getting result data

•False: perform a normal get result data (default for 6.2)

•Keeping False by default for now until the columnset's are properly identified in the server export

sse_format : str, optional

•default: 'xml_obj'

•format to have server side export report in, one of: {'csv', 'xml', 'xml_obj', 'cef', 0, 1, 2}

leading : str, optional

•default: ''

•used for sse_format 'cef' only, the string to prepend to each row

trailing : str, optional

•default: ''

•used for sse_format 'cef' only, the string to append to each row

Returnsret : dict, containing

•*question_object* : `taniumpy.object_types.saved_question.SavedQuestion`, the saved question object

•*question_object* : `taniumpy.object_types.question.Question`, the question asked by *saved_question_object*

•*question_results* : `taniumpy.object_types.result_set.ResultSet`, the results for *question_object*

•*poller_object* : None if *refresh_data* == False, elsewise `pytan.pollers.QuestionPoller`, poller object used to wait until all results are in before getting *question_results*,

•*poller_success* : None if *refresh_data* == False, elsewise True or False

Notes

id or name must be supplied

create_dashboard(*name*, *text=''*, *group=''*, *public_flag=True*, ***kwargs*)

Calls `pytan.handler.Handler.run_plugin()` to run the CreateDashboard plugin and parse the response

Parametersname : str

•name of dashboard to create

text : str, optional

•default: ''

•text for this dashboard

group : str, optional

•default: ''

•group name for this dashboard

public_flag : bool, optional

•default: True

•True: make this dashboard public

•False: do not make this dashboard public

Returnsplugin_result, sql_zipped : tuple

- plugin_result will be the taniumpy object representation of the SOAP response from Tanium server
- sql_zipped will be a dict with the SQL results embedded in the SOAP response

create_from_json(*objtype*, *json_file*, ***kwargs*)

Creates a new object using the SOAP api from a json file

Parametersobjtype : str

- Type of object described in *json_file*

json_file : str

- path to JSON file that describes an API object

Returnsret : *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

create_group(*groupname*, *filters=[]*, *filter_options=[]*, ***kwargs*)

Create a group object

Parametersgroupname : str

- name of group to create

filters : str or list of str, optional

- default: []
- each string must describe a filter

filter_options : str or list of str, optional

- default: []
- each string must describe an option for *filters*

filters_help : bool, optional

- default: False
- False: do not print the help string for filters
- True: print the help string for filters and exit

options_help : bool, optional

- default: False
- False: do not print the help string for options
- True: print the help string for options and exit

Returnsgroup_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_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*, ***kwargs*)

Create a package object

Parametersname : 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

•default: []

•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

•default: 600

•timeout for command execution in seconds

parameters_json_file : str, optional

•default: ''

•path to json file describing parameters for package

expire_seconds : int, optional

•default: 600

•timeout for action expiry in seconds

verify_filters : str or list of str, optional

•default: []

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

verify_filter_options : str or list of str, optional

•default: []

•each string must describe an option for *verify_filters*

verify_expire_seconds : int, optional

•default: 600

•timeout for verify action expiry in seconds

filters_help : bool, optional

•default: False

•False: do not print the help string for filters

•True: print the help string for filters and exit

options_help : bool, optional

•default: False

•False: do not print the help string for options

•True: print the help string for options and exit

metadata: list of list of strs, optional

•default: []

•each list must be a 2 item list:

•list item 1 property name

•list item 2 property value

Returnspackage_obj : *taniumpy.object_types.package_spec.PackageSpec*

•TaniumPy object added to Tanium SOAP Server

See also:

pytan.constants.FILTER_MAPSvalid filters for verify_filters

pytan.constants.OPTION_MAPSvalid options for verify_filter_options

create_report_file(*contents*, *report_file=None*, ***kwargs*)

Exports a python API object to a file

Parameterscontents : str

•contents to write to *report_file*

report_file : str, optional

•filename to save report as

report_dir : str, optional

•default: None

•directory to save report in, will use current working directory if not supplied

prefix : str, optional

•default: ‘’

•prefix to add to *report_file*

postfix : str, optional

•default: ‘’

•postfix to add to *report_file*

Returnsreport_path : str

•the full path to the file created with *contents*

create_sensor(***kwargs*)

Create a sensor object

Raisespytan.exceptions.HandlerError : `pytan.utils.pytan.exceptions.HandlerError`

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

create_user(*name*, *rolename=[]*, *roleid=[]*, *properties=[]*, ***kwargs*)

Create a user object

Parametersname : str

•name of user to create

rolename : str or list of str, optional

•default: []

•name(s) of roles to add to user

roleid : int or list of int, optional

•default: []

•id(s) of roles to add to user

properties: list of list of strs, optional

•default: []

•each list must be a 2 item list:

•list item 1 property name

•list item 2 property value

Returnsuser_obj : `taniumpy.object_types.user.User`

•TaniumPy object added to Tanium SOAP Server

create_whitelisted_url(*url*, *regex=False*, *download_seconds=86400*, *properties=[]*, ***kwargs*)

Create a whitelisted url object

Parametersurl : str

•text of new url

regex : bool, optional

•default: False

•False: *url* is not a regex pattern

•True: *url* is a regex pattern

download_seconds : int, optional

•default: 86400

•how often to re-download *url*

properties: list of list of strs, optional

•default: []

•each list must be a 2 item list:

•list item 1 property name

•list item 2 property value

Returnsurl_obj : `taniumpy.object_types.white_listed_url.WhiteListedUrl`

•TaniumPy object added to Tanium SOAP Server

delete(*objtype*, ***kwargs*)
Delete an object type

Parametersobjtype : 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

Returnsret : dict

•dict containing deploy action object and results from deploy action

See also:

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

delete_dashboard(*name*, ***kwargs*)
Calls *pytan.handler.Handler.run_plugin()* to run the DeleteDashboards plugin and parse the response

Parametersname : str

•name of dashboard to delete

Returnsplugin_result, sql_zipped : tuple

•plugin_result will be the taniumpy object representation of the SOAP response from Tanium server

•sql_zipped will be a dict with the SQL results embedded in the SOAP response

deploy_action(***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()*

Parameterspackage : str

•package to deploy with this action

action_filters : str, list of str, optional

•default: []

•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

•default: []

•options to apply to *action_filters*

start_seconds_from_now : int, optional

•default: 0

•start action N seconds from now

expire_seconds : int, optional

•default: package.expire_seconds

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

run : bool, optional

•default: False

•False: just ask the question that pertains to verify action, export the results to CSV, and raise pytan.exceptions.RunFalse – does not deploy the action

•True: actually deploy the action

get_results : bool, optional

•default: True

•True: wait for result completion after deploying action

•False: just deploy the action and return the object in *ret*

package_help : bool, optional

•default: False

•False: do not print the help string for package

•True: print the help string for package and exit

filters_help : bool, optional

•default: False

•False: do not print the help string for filters

•True: print the help string for filters and exit

options_help : bool, optional

•default: False

•False: do not print the help string for options

•True: print the help string for options and exit

Returnsret : dict, containing:

•*saved_action_object* : `taniumpy.object_types.saved_action.SavedAction`, the saved_action added for this action (None if 6.2)

•*action_object* : `taniumpy.object_types.action.Action`, the action object that tanium created for *saved_action*

•*package_object* : `taniumpy.object_types.package_spec.PackageSPec`, the package object used in *saved_action*

•*action_info* : `taniumpy.object_types.result_info.ResultInfo`, the initial GetResultInfo call done before getting results

•*poller_object* : `pytan.pollers.ActionPoller`, poller object used to wait until all results are in before getting *action_results*

•*poller_success* : None if *get_results* == False, elsewise True or False

•*action_results* : None if *get_results* == False, elsewise `taniumpy.object_types.result_set.ResultSet`, the results for *action_object*

•*action_result_map* : None if *get_results* == False, elsewise progress map for *action_object* in dictionary form

See also:

pytan.constants.FILTER_MAPS valid filter dictionaries for filters

pytan.constants.OPTION_MAPS valid option dictionaries for options

pytan.handler.Handler._deploy_action() private method with the actual workflow used to create and add the action object

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']
```

export_obj(*obj*, *export_format='csv'*, ***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, optional

- default: 'csv'
- the format to export *obj* to, one of: {'csv', 'xml', 'json'}

header_sort : list of str, bool, optional

- default: True
- 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

- default: False
- 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

- default: False

•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

•default: False

•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

•default: False

•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

•default: False

•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

Returnsresult : str

•the contents of exporting *export_format*

See also:

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

Notes

When performing a CSV export and importing that CSV into excel, keep in mind that Excel has a per cell character limit of 32,000. Any cell larger than that will be broken up into a whole new row, which can wreak havoc with data in Excel.

export_to_report_file(*obj*, *export_format='csv'*, ***kwargs*)

Exports a python API object to a file

Parametersobj : *taniumpy.object_types.base.BaseType* or *taniumpy.object_types.result_set.ResultSet*

•TaniumPy object to export

export_format : str, optional

•default: 'csv'

•the format to export *obj* to, one of: {'csv', 'xml', 'json'}

header_sort : list of str, bool, optional

•default: True

•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

•default: False

•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

•default: False

•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

•default: False

•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

•default: False

•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

•default: False

•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

•default: None

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

report_dir: str, optional

•default: None

•directory to save report in, will use current working directory if not supplied

prefix: str, optional

•default: ‘’

•prefix to add to *report_file*

postfix: str, optional

•default: ‘’

•postfix to add to *report_file*

Returnsreport_path, result : tuple

•report_path : str, the full path to the file created with contents of *result*

•result : str, the contents written to report_path

See also:

pytan.handler.Handler.export_obj() method that performs the actual work to do the exporting

pytan.handler.Handler.create_report_file() method that performs the actual work to write the report file

Notes

When performing a CSV export and importing that CSV into excel, keep in mind that Excel has a per cell character limit of 32,000. Any cell larger than that will be broken up into a whole new row, which can wreak havoc with data in Excel.

get (*objtype*, ***kwargs*)
Get an object type

Parametersobjtype : 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

Returnsobj_list : *taniumpy.object_types.base.BaseType*

•The object list of items found for *objtype*

See also:

pytan.constants.GET_OBJ_MAP maps objtype to supported ‘search’ keys

pytan.handler.Handler._get_multi() private method used to get multiple items

pytan.handler.Handler._get_single() private method used to get singular items

get_all (*objtype*, ***kwargs*)
Get all objects of a type

Parametersobjtype : string

•type of object to get

Returnsobj_list : *taniumpy.object_types.base.BaseType*

•The object list of items found for *objtype*

See also:

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

pytan.handler.Handler._find() private method used to find items

get_dashboards (*name=''*, ***kwargs*)
Calls *pytan.handler.Handler.run_plugin()* to run the GetDashboards plugin and parse the response

Parametersname : str, optional

•default: ''

•name of dashboard to get, if empty will return all dashboards

Returnsplugin_result, sql_zipped : tuple

•plugin_result will be the taniumpy object representation of the SOAP response from Tanium server

•sql_zipped will be a dict with the SQL results embedded in the SOAP response

get_result_data (*obj*, *aggregate=False*, *shrink=True*, ***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*.

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to get result data for

aggregate : bool, optional

•default: False

•False: get all the data

•True: get just the aggregate data (row counts of matches)

shrink : bool, optional

•default: True

•True: Shrink the object down to just id/name/hash attributes (for smaller request)

•False: Use the full object as is

Returnsrd : *taniumpy.object_types.result_set.ResultSet*

The return of GetResultData for *obj*

get_result_data_sse (*obj*, *sse_format='csv'*, *leading=''*, *trailing=''*, ***kwargs*)
Get the result data for a python API object using a server side export (sse)

This method issues a GetResultData command to the SOAP api for *obj* with the option *export_flag* set to 1. This will cause the server to process all of the data for a given result set and save it as *export_format*. Then the user can use an authenticated GET request to get the status of the file via "/export/${export_id}.status". Once the status returns "Completed.", the actual report file can be retrieved by an authenticated GET request to "/export/${export_id}.gz". This workflow saves a lot of processing time and removes the need to paginate large result sets necessary in normal GetResultData calls.

Version support

•6.5.314.4231: initial sse support (csv only)

•6.5.314.4300: export_format support (adds xml and cef)

•6.5.314.4300: fix core dump if multiple sse done on empty resultset

•6.5.314.4300: fix no status file if sse done on empty resultset

•6.5.314.4300: fix response if more than two sse done in same second

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to get result data for

sse_format : str, optional

•default: 'csv'

•format to have server create report in, one of: {'csv', 'xml', 'xml_obj', 'cef', 0, 1, 2}

leading : str, optional

•default: ''

•used for sse_format 'cef' only, the string to prepend to each row

trailing : str, optional

•default: ''

•used for sse_format 'cef' only, the string to append to each row

Returnsexport_data : either *str* or *taniumpy.object_types.result_set.ResultSet*

•If sse_format is one of csv, xml, or cef, export_data will be a *str* containing the contents of the ResultSet in said format

•If sse_format is xml_obj, export_data will be a *taniumpy.object_types.result_set.ResultSet*

See also:

pytan.constants.SSE_FORMAT_MAPmaps *sse_format* to an integer for use by the SOAP API

pytan.constants.SSE_RESTRICT_MAPmaps sse_format integers to supported platform versions

pytan.constants.SSE_CRASH_MAPmaps platform versions that can cause issues in various scenarios

get_result_info(*obj*, *shrink=True*, ***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.

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to get result data for

shrink : bool, optional

•default: True

•True: Shrink the object down to just id/name/hash attributes (for smaller request)

•False: Use the full object as is

Returnsri : *taniumpy.object_types.result_info.ResultInfo*

•The return of GetResultData for *obj*

get_server_version (***kwargs*)
Uses `taniumpy.session.Session.get_server_version()` to get the version of the Tanium Server

Returnsserver_version: str

•Version of Tanium Server in string format

parse_query (*question_text*, ***kwargs*)
Ask a parsed question as *question_text* and get a list of parsed results back

Parametersquestion_text : str

•The question text you want the server to parse into a list of parsed results

Returnsparse_job_results : *taniumpy.object_types.parse_result_group.ParseResultGroup*

run_plugin (*obj*, ***kwargs*)
Wrapper around `pytan.session.Session.run_plugin()` to run the plugin and zip up the SQL results into a python dictionary

Parametersobj : *taniumpy.object_types.plugin.Plugin*

•Plugin object to run

Returnsplugin_result, sql_zipped : tuple

•plugin_result will be the taniumpy object representation of the SOAP response from Tanium server

•sql_zipped will be a dict with the SQL results embedded in the SOAP response

stop_action (*id*, ***kwargs*)
Stop an action

Parametersid : int

•id of action to stop

Returnsaction_stop_obj : *taniumpy.object_types.action_stop.ActionStop*

The object containing the ID of the action stop job

xml_to_result_set_obj (*x*, ***kwargs*)
Wraps a Result Set XML from a server side export in the appropriate tags and returns a ResultSet object

Parametersx : str

•str of XML to convert to a ResultSet object

Returnsrs : *taniumpy.object_types.result_set.ResultSet*

•x converted into a ResultSet object

1.2.2 pytan.sessions

Session classes for the *pytan* module.

class `pytan.sessions.`**Session** (*host*, *port=443*, ***kwargs*)
Bases: `object`

This session object uses the *requests* package instead of the built in httplib library.

This provides support for keep alive, gzip, cookies, forwarding, and a host of other features automatically.

Examples

Setup a Session() object:

```
>>> import sys
>>> sys.path.append('/path/to/pytan/')
>>> import pytan
>>> session = pytan.sessions.Session('host')
```

Authenticate with the Session() object:

```
>>> session.authenticate('username', 'password')
```

ALL_REQUESTS_RESPONSES = []
This list will be updated with each requests response object that was received

AUTH_CONNECT_TIMEOUT_SEC = 5
number of seconds before timing out for a connection while authenticating

AUTH_FAIL_CODES = [401, 403]
List of HTTP response codes that equate to authorization failures

AUTH_RES = 'auth'
The URL to use for authentication requests

AUTH_RESPONSE_TIMEOUT_SEC = 15
number of seconds before timing out for a response while authenticating

BAD_RESPONSE_CMD_PRUNES = ['\n', 'XML Parse Error: ', 'SOAPProcessing Exception: class ', 'ERROR: 400 Bad Re
List of strings to remove from commands in responses that do not match the response in the request

BAD_SERVER_VERSIONS = [None, '', 'Unable to determine', 'Not yet determined']
List of server versions that are not valid

ELEMENT_RE_TXT = '<{0}>(.*?)</{0}>'
regex string to search for an element in XML bodies

HTTP_AUTH_RETRY = True
retry HTTP GET/POST's with username/password if session_id fails or not

HTTP_DEBUG = False
print requests package debug or not

HTTP_RETRY_COUNT = 5
number of times to retry HTTP GET/POST's if the connection times out/fails

INFO_CONNECT_TIMEOUT_SEC = 5
number of seconds before timing out for a connection while getting server info

INFO_RES = 'info.json'
The URL to use for server info requests

INFO_RESPONSE_TIMEOUT_SEC = 15
number of seconds before timing out for a response while getting server info

LAST_REQUESTS_RESPONSE = None
This variable will be updated with the last requests response object that was received

LAST_RESPONSE_INFO = {}
This variable will be updated with the information from the most recent call to _get_response()

RECORD_ALL_REQUESTS = False
Controls whether each requests response object is appended to the self.ALL_REQUESTS_RESPONSES list

REQUESTS_SESSION = None
The Requests session allows you to persist certain parameters across requests. It also persists cookies across all requests made from the Session instance. Any requests that you make within a session will automatically reuse the appropriate connection

REQUEST_BODY_BASE = '<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmln
The XML template used for all SOAP Requests in string form

SOAP_CONNECT_TIMEOUT_SEC = 15
number of seconds before timing out for a connection while sending a SOAP Request

SOAP_REQUEST_HEADERS = {'Content-Type': 'text/xml; charset=utf-8', 'Accept-Encoding': 'gzip'}
dictionary of headers to add to every HTTP GET/POST

SOAP_RES = 'soap'
The URL to use for SOAP requests

SOAP_RESPONSE_TIMEOUT_SEC = 540
number of seconds before timing out for a response while sending a SOAP request

STATS_LOOP_ENABLED = False
enable the statistics loop thread or not

STATS_LOOP_SLEEP_SEC = 5
number of seconds to sleep in between printing the statistics when stats_loop_enabled is True

STATS_LOOP_TARGETS = [{'Version': 'Settings/Version'}, {'Active Questions': 'Active Question Cache/Active Question
list of dictionaries with the key being the section of info.json to print info from, and the value being the item with in that section to print the value

XMLNS = {'xsi': 'xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"', 'typens': 'xmlns:typens="urn:TaniumSOA
The namespace mappings for use in XML Request bodies

_build_body(*command*, *object_list*, *log_options=False*, ***kwargs*)
Utility method for building an XML Request Body

Parameterscommand : str

- text to use in command node when building template

object_list : str

- XML string to use in object list node when building template

kwargs : dict, optional

- any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

log_options : bool, optional

- default: False
- False: Do not print messages setting attributes in Options from keys in kwargs
- True: Print messages setting attributes in Options from keys in kwargs

Returnsbody : str

- The XML request body created from the string.template self.REQUEST_BODY_TEMPLATE

_check_auth()
Utility method to check if authentication has been done yet, and throw an exception if not

_clean_headers(*headers=None*)
Utility method for getting the headers for the current request, combining them with the session headers used for every request, and obfuscating the value of any 'password' header.

Parametersheaders : dict

•dict of key/value pairs for a set of headers for a given request

Returnsheaders : dict

•dict of key/value pairs for a set of cleaned headers for a given request

_create_add_object_body(*obj*, ***kwargs*)
Utility method for building an XML Request Body to add an object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to convert into XML

kwargs : dict, optional

•any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

Returnsobj_body : str

•The XML request body created from *pytan.sessions.Session._build_body()*

_create_delete_object_body(*obj*, ***kwargs*)
Utility method for building an XML Request Body to delete an object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to convert into XML

kwargs : dict, optional

•any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

Returnsobj_body : str

•The XML request body created from *pytan.sessions.Session._build_body()*

_create_get_object_body(*obj*, ***kwargs*)
Utility method for building an XML Request Body to get an object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to convert into XML

kwargs : dict, optional

•any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

Returnsobj_body : str

•The XML request body created from *pytan.sessions.Session._build_body()*

_create_get_result_data_body(*obj*, ***kwargs*)
Utility method for building an XML Request Body to get result data for an object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to convert into XML

kwargs : dict, optional

•any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

Returnsobj_body : str

•The XML request body created from *pytan.sessions.Session._build_body()*

_create_get_result_info_body(*obj*, ***kwargs*)
Utility method for building an XML Request Body to get result info for an object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to convert into XML

kwargs : dict, optional

•any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

Returnsobj_body : str

•The XML request body created from *pytan.sessions.Session._build_body()*

_create_run_plugin_object_body(*obj*, ***kwargs*)
Utility method for building an XML Request Body to run a plugin

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to convert into XML

kwargs : dict, optional

•any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

Returnsobj_body : str

•The XML request body created from *pytan.sessions.Session._build_body()*

_create_update_object_body(*obj*, ***kwargs*)
Utility method for building an XML Request Body to update an object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to convert into XML

kwargs : dict, optional

•any number of attributes that can be set via *taniumpy.object_types.options.Options* that control the servers response.

Returnsobj_body : str

•The XML request body created from *pytan.sessions.Session._build_body()*

_extract_resultxml(*response_body*)
Utility method to get the 'ResultXML' element from an XML body

Parametersresponse_body : str

•XML body to search for the 'ResultXML' element in

Returnsret : str of ResultXML element

•str if 'export_id' element found in XML

_find_stat_target(*target*, *diags*)
Utility method for finding a target in info.json and returning the value, optionally performing a percentage calculation on two values if the target[0] starts with percentage(

Parameterstarget : list

•index0 : label : human friendly name to refer to search_path

•index1 : search_path : / seperated search path to find a given value from info.json

diags : dict

•flattened dictionary of info.json diagnostics

Returnsdict

•label : same as provided in *target* index0 (label)

•result : value resolved from *pytan.sessions.Session._resolve_stat_target()* for *target* index1 (search_path)

_flatten_server_info(*structure*)
Utility method for flattening the JSON structure for info.json into a more python usable format

Parametersstructure

•dict/tuple/list to flatten

Returnsflattened

•the dict/tuple/list flattened out

_full_url(*url*, ***kwargs*)
Utility method for constructing a full url

Parametersurl : str

•url to use in string

host : str, optional

•default: self.host

•hostname/IP address to use in string

port : str, optional

•default: self.port

•port to use in string

Returnsfull_url : str

•full url in the form of https://$host:$port/$url

_get_percentage(*part*, *whole*)
Utility method for getting percentage of part out of whole

Parameterspart: int, float

whole: int, float

Returnsstr : the percentage of part out of whole in 2 decimal places

_get_response(*request_body*, ***kwargs*)
This is a wrapper around *pytan.sessions.Session.http_post()* for SOAP XML requests and responses.

This method will update self.session_id if the response contains a different session_id than what is currently in this object.

Parametersrequest_body : str

•the XML request body to send to the server

connect_timeout: int, optional

•default: self.SOAP_CONNECT_TIMEOUT_SEC

•timeout in seconds for connection to host

response_timeout: int, optional

•default: self.SOAP_RESPONSE_TIMEOUT_SEC

•timeout in seconds for response from host

retry_auth: bool, optional

•default: True

•True: retry authentication with username/password if session_id fails

•False: throw exception if session_id fails

retry_count: int, optional

•number of times to retry the request if the server fails to respond properly or in time

pytan_help : str, optional

•default: ‘’

•help string to add to self.LAST_REQUESTS_RESPONSE.pytan_help

Returnsbody : str

•str containing body of response from server

See also:

pytan.sessions.Session.http_post() wrapper method used to perform the HTTP POST

_http_get(*host*, *port*, *url*, *headers=None*, *connect_timeout=15*, *response_timeout=180*, *debug=False*, *pytan_help=’‘*, ***kwargs*)

This is an HTTP GET method that utilizes the *requests* package.

Parametershost : str

•host to connect to

port : int

•port to connect to

url : str

•url to fetch on the server

headers : dict, optional

•default: None

•headers to supply as part of POST request

connect_timeout : int, optional

•default: 15

•timeout in seconds for connection to host

response_timeout : int, optional

•default: 180

•timeout in seconds for response from host

debug : bool, optional

•default: False

•False: do not print requests debug messages

•True: print requests debug messages

pytan_help : str, optional

•default: ''

•help string to add to self.LAST_REQUESTS_RESPONSE.pytan_help

perform_xml_clean : bool, optional

•default: False

•False: Do not run the response_body through an XML cleaner

•True: Run the response_body through an XML cleaner before returning it

clean_restricted : bool, optional

•default: True

•True: When XML cleaning the response_body, remove restricted characters as well as invalid characters

•False: When XML cleaning the response_body, remove only invalid characters

log_clean_messages : bool, optional

•default: True

•True: When XML cleaning the response_body, enable logging messages about invalid/restricted matches

•False: When XML cleaning the response_body, disable logging messages about invalid/restricted matches

log_bad_characters : bool, optional

•default: False

•False: When XML cleaning the response_body, disable logging messages about the actual characters that were invalid/restricted

•True: When XML cleaning the response_body, enable logging messages about the actual characters that were invalid/restricted

Returnsbody : str

•str containing body of response from server

_http_post(*host*, *port*, *url*, *body=None*, *headers=None*, *connect_timeout=15*, *response_timeout=180*, *debug=False*, *pytan_help=''*, ***kwargs*)

This is an HTTP POST method that utilizes the `requests` package.

Parametershost : str

•host to connect to

port : int

- port to connect to

url : str

- url to fetch on the server

body : str, optional

- default: None
- body to send as part of the POST request

headers : dict, optional

- default: None
- headers to supply as part of POST request

connect_timeout : int, optional

- default: 15
- timeout in seconds for connection to host

response_timeout : int, optional

- default: 180
- timeout in seconds for response from host

debug : bool, optional

- default: False
- False: do not print requests debug messages
- True: print requests debug messages

pytan_help : str, optional

- default: ''
- help string to add to self.LAST_REQUESTS_RESPONSE.pytan_help

perform_xml_clean : bool, optional

- default: True
- True: Run the response_body through an XML cleaner before returning it
- False: Do not run the response_body through an XML cleaner

clean_restricted : bool, optional

- default: True
- True: When XML cleaning the response_body, remove restricted characters as well as invalid characters
- False: When XML cleaning the response_body, remove only invalid characters

log_clean_messages : bool, optional

- default: True
- True: When XML cleaning the response_body, enable logging messages about invalid/restricted matches

•False: When XML cleaning the response_body, disable logging messages about invalid/restricted matches

log_bad_characters : bool, optional

•default: False

•False: When XML cleaning the response_body, disable logging messages about the actual characters that were invalid/restricted

•True: When XML cleaning the response_body, enable logging messages about the actual characters that were invalid/restricted

Returnsbody : str

•str containing body of response from server

See also:

pytan.xml_clean.xml_cleaner() function to remove invalid/bad characters from XML responses

_invalid_server_version()
Utility method to find out if self.server_version is valid or not

_regex_body_for_element(*body*, *element*, *fail=True*)
Utility method to use a regex to get an element from an XML body

Parametersbody : str

•XML to search

element : str

•element name to search for in body

fail : bool, optional

•default: True

•True: throw exception if unable to find any matches for *regex* in *body*

•False do not throw exception if unable to find any matches for *regex* in *body*

Returnsret : str

•The first value that matches the regex ELEMENT_RE_TXT with element

Notes

•Using regex is WAY faster than ElementTree chewing the body in and out, this matters a LOT on LARGE return bodies

_replace_auth(*headers*)
Utility method for removing username, password, and/or session from supplied headers and replacing them with the current objects session or username and password

Parametersheaders : dict

•dict of key/value pairs for a set of headers for a given request

Returnsheaders : dict

•dict of key/value pairs for a set of headers for a given request

_resolve_stat_target(*search_path*, *diags*)
Utility method for resolving the value of search_path in info.json and returning the value

Parameterssearch_path : str

•/ seperated search path to find a given value from info.json

diags : dict

•flattened dictionary of info.json diagnostics

Returnsstr

•value resolved from *diags* for *search_path*

_start_stats_thread(***kwargs*)
Utility method starting the *pytan.sessions.Session._stats_loop()* method in a threaded daemon

_stats_loop(***kwargs*)
Utility method for logging server stats via *pytan.sessions.Session.get_server_stats()* every self.STATS_LOOP_SLEEP_SEC

add(*obj*, ***kwargs*)
Creates and sends a AddObject XML Request body from *obj* and parses the response into an appropriate *taniumpy* object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to add

Returnsobj : *taniumpy.object_types.base.BaseType*

•added object

authenticate(*username=None*, *password=None*, *session_id=None*, ***kwargs*)
Authenticate against a Tanium Server using a username/password or a session ID

Parametersusername : str, optional

•default: None

•username to authenticate as

password : str, optional

•default: None

•password for *username*

session_id : str, optional

•default: None

•session_id to authenticate with, this will be used in favor of username/password if all 3 are supplied.

persistent: bool, optional

•default: False

•False: do not request a persistent session (returns a session_id that expires 5 minutes after last use)

•True: do request a persistent (returns a session_id that expires 1 week after last use)

pytan_help : str, optional

•default: ‘’

•help string to add to self.LAST_REQUESTS_RESPONSE.pytan_help

Notes

Can request a persistent session that will last up to 1 week when authenticating with username and password.

New persistent sessions may be handed out by the Tanium server when the session handed by this auth call is used to login with that week. The new session must be used to login, as no matter what persistent sessions will expire 1 week after issuance (or when logout is called with that session, or when logout with all_sessions=True is called for any session for this user)

the way sessions get issued:

- •a GET request to /auth is issued
- •username/password supplied in headers as base64 encoded, or session is supplied in headers as string
- •session is returned upon successful auth
- •if there is a header “persistent=1” in the headers, a session that expires after 1 week will be issued if username/password was used to auth. persistent is ignored if session is used to auth.
- •if there is not a header “persistent=1” in the headers, a session that expires after 5 minutes will be issued
- •if session is used before it expires, it’s expiry will be extended by 5 minutes or 1 week, depending on the type of persistence
- •while using the SOAP api, new session ID’s may be returned as part of the response. these new session ID’s should be used in lieu of the old session ID

/auth URL This url is used for validating a server user’s credentials. It supports a few different ways to authenticate and returns a SOAP session ID on success. These sessions expire after 5 minutes by default if they aren’t used in SOAP requests. This expiration is configured with the server setting ‘session_expiration_seconds’.

Supported Authentication Methods:

- •HTTP Basic Auth (Clear Text/BASE64)
- •Username/Password/Domain Headers (Clear Text)
- •Negotiate (NTLM Only)

NTLM is enabled by default in 6.3 or greater and requires a persistent connection until a session is generated.

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

Creates and sends a DeleteObject XML Request body from *obj* and parses the response into an appropriate *taniumpy* object

Parametersobj : *taniumpy.object_types.base.BaseType*

- •object to delete

Returnsobj : *taniumpy.object_types.base.BaseType*

- •deleted object

disable_stats_loop(*sleep=None*)
Disables the stats loop thread, which will print out the results of *pytan.sessions.Session.get_server_stats()* every *pytan.sessions.Session.STATS_LOOP_SLEEP_SEC*

Parameters
sleep : int, optional

- when disabling the stats loop, update *pytan.sessions.Session.STATS_LOOP_SLEEP_SEC* with *sleep*

See also:

pytan.sessions.Session._stats_loop() method started as a thread which checks self.STATS_LOOP_ENABLED before running *pytan.sessions.Session.get_server_stats()*

enable_stats_loop(*sleep=None*)
Enables the stats loop thread, which will print out the results of *pytan.sessions.Session.get_server_stats()* every *pytan.sessions.Session.STATS_LOOP_SLEEP_SEC*

Parameters
sleep : int, optional

- when enabling the stats loop, update *pytan.sessions.Session.STATS_LOOP_SLEEP_SEC* with *sleep*

See also:

pytan.sessions.Session._stats_loop() method started as a thread which checks self.STATS_LOOP_ENABLED before running *pytan.sessions.Session.get_server_stats()*

find(*obj*, ***kwargs*)
Creates and sends a GetObject XML Request body from *object_type* and parses the response into an appropriate *taniumpy* object

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

- object to find

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

- found objects

get_result_data(*obj*, ***kwargs*)
Creates and sends a GetResultData XML Request body from *obj* and parses the response into an appropriate *taniumpy* object

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

- object to get result set for

Returns
obj : *taniumpy.object_types.result_set.ResultSet*

- otherwise, *obj* will be the ResultSet for *obj*

get_result_data_sse(*obj*, ***kwargs*)
Creates and sends a GetResultData XML Request body that starts a server side export from *obj* and parses the response for an export_id.

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

- object to start server side export

Returnsexport_id : str

•value of export_id element found in response

get_result_info(*obj*, ***kwargs*)

Creates and sends a GetResultInfo XML Request body from *obj* and parses the response into an appropriate *taniumpy* object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to get result info for

Returnsobj : *taniumpy.object_types.result_info.ResultInfo*

•ResultInfo for *obj*

get_server_info(*port=None*, *fallback_port=444*, ***kwargs*)

Gets the /info.json

Parametersport : int, optional

•default: None

•port to attempt getting /info.json from, if not specified will use self.port

fallback_port : int, optional

•default: 444

•fallback port to attempt getting /info.json from if *port* fails

Returnsinfo_dict : dict

•raw json response converted into python dict

•'diags_flat': info.json flattened out into an easier to use structure for python handling

•'server_info_pass_msgs': messages about successfully retrieving info.json

•'server_info_fail_msgs': messages about failing to retrieve info.json

See also:

pytan.sessions.Session._flatten_server_info() method to flatten the dictionary received from info.json into a python friendly format

Notes

•6.2 /info.json is only available on soap port (default port: 444)

•6.5 /info.json is only available on server port (default port: 443)

get_server_stats(***kwargs*)

Creates a str containing a number of stats gathered from /info.json

Returnsstr

•str containing stats from /info.json

See also:

pytan.sessions.Session.STATS_LOOP_TARGETS list of dict containing stat keys to pull from /info.json

get_server_version(***kwargs*)
Tries to parse the server version from /info.json

Returnsstr

•str containing server version from /info.json

host = None
host to connect to

http_get(*url*, ***kwargs*)
This is an authenticated HTTP GET method. It will always forcibly use the authentication credentials that are stored in the current object when performing an HTTP GET.

Parametersurl : str

•url to fetch on the server

host : str, optional

•default: self.host

•host to connect to

port : int, optional

•default: self.port

•port to connect to

headers : dict, optional

•default: {}

•headers to supply as part of GET request

connect_timeout : int, optional

•default: self.SOAP_CONNECT_TIMEOUT_SEC

•timeout in seconds for connection to host

response_timeout : int, optional

•default: self.SOAP_RESPONSE_TIMEOUT_SEC

•timeout in seconds for response from host

debug : bool, optional

•default: self.HTTP_DEBUG

•False: do not print requests debug messages

•True: print requests debug messages

auth_retry : bool, optional

•default: self.HTTP_AUTH_RETRY

•True: retry authentication with username/password if session_id fails

•False: throw exception if session_id fails

retry_count : int, optional

•default: self.HTTP_RETRY_COUNT

•number of times to retry the GET request if the server fails to respond properly or in time

pytan_help : str, optional

•default: ‘’

•help string to add to self.LAST_REQUESTS_RESPONSE.pytan_help

Returnsbody : str

•str containing body of response from server

See also:

pytan.sessions.Session._http_get() private method used to perform the actual HTTP GET

http_post(***kwargs*)

This is an authenticated HTTP POST method. It will always forcibly use the authentication credentials that are stored in the current object when performing an HTTP POST.

Parametersurl : str, optional

•default: self.SOAP_RES

•url to fetch on the server

host : str, optional

•default: self.host

•host to connect to

port : int, optional

•default: self.port

•port to connect to

headers : dict, optional

•default: {}

•headers to supply as part of POST request

body : str, optional

•default: ‘’

•body to send as part of the POST request

connect_timeout : int, optional

•default: self.SOAP_CONNECT_TIMEOUT_SEC

•timeout in seconds for connection to host

response_timeout : int, optional

•default: self.SOAP_RESPONSE_TIMEOUT_SEC

•timeout in seconds for response from host

debug : bool, optional

•default: self.HTTP_DEBUG

•False: do not print requests debug messages

•True: print requests debug messages

auth_retry : bool, optional

•default: self.HTTP_AUTH_RETRY

•True: retry authentication with username/password if session_id fails

•False: throw exception if session_id fails

retry_count : int, optional

•default: self.HTTP_RETRY_COUNT

•number of times to retry the POST request if the server fails to respond properly or in time

pytan_help : str, optional

•default: ‘’

•help string to add to self.LAST_REQUESTS_RESPONSE.pytan_help

Returnsbody : str

•str containing body of response from server

See also:

pytan.sessions.Session._http_post() private method used to perform the actual HTTP POST

is_auth

Property to determine if there is a valid session_id or username and password stored in this object

Returnsbool

•True: if self._session_id or self._username and _self.password are set

•False: if not

logout(*all_session_ids=False*, ***kwargs*)

Logout a given session_id from Tanium. If not session_id currently set, it will authenticate to get one.

Parametersall_session_ids : bool, optional

•default: False

•False: only log out the current session id for the current user

•True: log out ALL session id’s associated for the current user

pytan_help : str, optional

•default: ‘’

•help string to add to self.LAST_REQUESTS_RESPONSE.pytan_help

platform_is_6_5(***kwargs*)

Check to see if self.server_version is less than 6.5

Returnsis6_5 : bool

•True if self.server_version is greater than or equal to 6.5

•False if self.server_version is less than 6.5

port = None

port to connect to

run_plugin(*obj*, ***kwargs*)

Creates and sends a RunPlugin XML Request body from *obj* and parses the response into an appropriate *taniumpy* object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to run

Returnsobj : *taniumpy.object_types.base.BaseType*

•results from running object

save(*obj*, ***kwargs*)
Creates and sends a UpdateObject XML Request body from *obj* and parses the response into an appropriate *taniumpy* object

Parametersobj : *taniumpy.object_types.base.BaseType*

•object to save

Returnsobj : *taniumpy.object_types.base.BaseType*

•saved object

server_version = 'Not yet determined'
version string of server, will be updated when get_server_version() is called

session_id
Property to fetch the session_id for this object

Returnsself._session_id : str

setup_logging()

1.2.3 pytan.pollers

Collection of classes and methods for polling of actions/questions in *pytan*

class pytan.pollers.**ActionPoller**(*handler*, *obj*, ***kwargs*)
Bases: *pytan.pollers.QuestionPoller*

A class to poll the progress of an Action.

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

•'SeenProgressChanged'

•'SeenAnswersComplete'

•'FinishedProgressChanged'

•'FinishedAnswersComplete'

Parametershandler : *pytan.handler.Handler*

•PyTan handler to use for GetResultInfo calls

obj : *taniumpy.object_types.action.Action*

•object to poll for progress

polling_secs : int, optional

•default: 5

•Number of seconds to wait in between GetResultInfo loops

complete_pct : int/float, optional

•default: 100

•Percentage of mr_tested out of estimated_total to consider the question "done"

override_timeout_secs : int, optional

•default: 0

•If supplied and not 0, timeout in seconds instead of when object expires

ACTION_DONE_KEY = 'success'
key in action_result_map that maps to an action being done

COMPLETE_PCT_DEFAULT = 100
default value for self.complete_pct

EXPIRATION_ATTR = 'expiration_time'
attribute of self.obj that contains the expiration for this object

OBJECT_TYPE
valid type of object that can be passed in as obj to __init__

alias of `Action`

RUNNING_STATUSES = ['active', 'open']
values for status attribute of action object that mean the action is running

_derive_object_info(***kwargs*)
Derive self.object_info from self.obj

_derive_package_spec(***kwargs*)
Get the package_spec attribute for self.obj, then fetch the full package_spec object

_derive_result_map(***kwargs*)
Determine what self.result_map should contain for the various statuses an action can have

A package object has to have a verify_group defined on it in order for deploy action verification to trigger. That can be only done at package creation/update

If verify_enable is True, then the various result states for an action change

_derive_status(***kwargs*)
Get the status attribute for self.obj

_derive_stopped_flag(***kwargs*)
Get the stopped_flag attribute for self.obj

_derive_target_group(***kwargs*)
Get the target_group attribute for self.obj, then fetch the full group object

_derive_verify_enabled(***kwargs*)
Determine if this action has verification enabled

_fix_group(*g*, ***kwargs*)
Sets ID to null on a group object and all of it's sub_groups, needed for 6.5

_post_init(***kwargs*)
Post init class setup

finished_eq_passed_loop(*callbacks={}*, ***kwargs*)
Method to poll Result Info for self.obj until the percentage of 'finished_count' out of 'self.passed_count' is greater than or equal to self.complete_pct

•finished_count is calculated from a full GetResultData call that is parsed into self.action_result_map

•self.passed_count is calculated by the question asked before this method is called. that question has no selects, but has a group that is the same group as the action for this object

run (*callbacks={}*, ***kwargs*)
Poll for action data and issue callbacks.

Parameterscallbacks : dict

- **Callbacks should be a dict with any of these members:**
  - 'SeenProgressChanged'
  - 'SeenAnswersComplete'
  - 'FinishedProgressChanged'
  - 'FinishedAnswersComplete'
- **Each callback should be a function that accepts:**
  - 'poller': a poller instance
  - 'pct': a percent complete
  - 'kwargs': a dict of other args

Notes

- Any callback can choose to get data from the session by calling `pytan.poller.QuestionPoller.get_result_data()` or new info by calling `pytan.poller.QuestionPoller.get_result_info()`
- Any callback can choose to stop the poller by calling `pytan.poller.QuestionPoller.stop()`
- Polling will be stopped only when one of the callbacks calls the `pytan.poller.QuestionPoller.stop()` method or the answers are complete.
- Any callbacks can call `pytan.poller.QuestionPoller.setPercentCompleteThreshold()` to change what "done" means on the fly

seen_eq_passed_loop (*callbacks={}*, ***kwargs*)
Method to poll Result Info for self.obj until the percentage of 'seen_count' out of 'self.passed_count' is greater than or equal to self.complete_pct

- seen_count is calculated from an aggregate GetResultData
- self.passed_count is calculated by the question asked before this method is called. that question has no selects, but has a group that is the same group as the action for this object

class `pytan.pollers.`**QuestionPoller** (*handler*, *obj*, ***kwargs*)
Bases: `object`

A class to poll the progress of a Question.

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

- ProgressChanged
- AnswersChanged
- AnswersComplete

Parametershandler : *pytan.handler.Handler*

- PyTan handler to use for GetResultInfo calls

obj : *taniumpy.object_types.question.Question*

•object to poll for progress

polling_secs : int, optional

•default: 5

•Number of seconds to wait in between GetResultInfo loops

complete_pct : int/float, optional

•default: 99

•Percentage of mr_tested out of estimated_total to consider the question "done"

override_timeout_secs : int, optional

•default: 0

•If supplied and not 0, timeout in seconds instead of when object expires

COMPLETE_PCT_DEFAULT = 99
default value for self.complete_pct

EXPIRATION_ATTR = 'expiration'
attribute of self.obj that contains the expiration for this object

EXPIRY_FALLBACK_SECS = 600
If the EXPIRATION_ATTR of *obj* can't be automatically determined, then this is used as a fallback for timeout - polling will failed after this many seconds if completion not reached

OBJECT_TYPE
valid type of object that can be passed in as obj to __init__

alias of `Question`

OVERRIDE_TIMEOUT_SECS_DEFAULT = 0
default value for self.override_timeout_secs

POLLING_SECS_DEFAULT = 5
default value for self.polling_secs

STR_ATTRS = ['object_info', 'polling_secs', 'override_timeout_secs', 'complete_pct', 'expiration']
Class attributes to include in __str__ output

_derive_attribute(*attr*, *fallback=''*, ***kwargs*)
Derive an attributes value from self.obj

Will re-fetch self.obj if the attribute is not set

Parametersattr : string

string of attribute name to fetch from self.obj

fallback : string

value to fallback to if it still can't be accessed after re-fetching the obj if fallback is None, an exception will be raised

Returnsval : perspective

The value of the attr from self.obj

_derive_expiration(***kwargs*)
Derive the expiration datetime string from a object

Will generate a datetime string from self.EXPIRY_FALLBACK_SECS if unable to get the expiration from the object (self.obj) itself.

_derive_object_info(***kwargs*)
Derive self.object_info from self.obj

_post_init(***kwargs*)
Post init class setup

_refetch_obj(***kwargs*)
Utility method to re-fetch a object

This is used in the case that the obj supplied does not have all the metadata available

_stop = False
Controls whether a run() loop should stop or not

get_result_data(***kwargs*)
Simple utility wrapper around *pytan.handler.Handler.get_result_data()*

get_result_info(***kwargs*)
Simple utility wrapper around *pytan.handler.Handler.get_result_info()*

handler = None
The Handler object for this poller

obj = None
The object for this poller

passed_eq_est_total_loop(*callbacks={}*, ***kwargs*)
Method to poll Result Info for self.obj until the percentage of 'passed' out of 'estimated_total' is greater than or equal to self.complete_pct

result_info = None
This will be updated with the ResultInfo object during run() calls

run(*callbacks={}*, ***kwargs*)
Poll for question data and issue callbacks.

Parameterscallbacks : dict

- **Callbacks should be a dict with any of these members:**
  - 'ProgressChanged'
  - 'AnswersChanged'
  - 'AnswersComplete'
- **Each callback should be a function that accepts:**
  - 'poller': a poller instance
  - 'pct': a percent complete
  - 'kwargs': a dict of other args

Notes

- Any callback can choose to get data from the session by calling poller.get_result_data() or new info by calling poller.get_result_info()
- Any callback can choose to stop the poller by calling poller.stop()
- Polling will be stopped only when one of the callbacks calls the stop() method or the answers are complete.

•Any callback can call setPercentCompleteThreshold to change what "done" means on the fly

run_callback(*callbacks*, *callback*, *pct*, ***kwargs*)
Utility method to find a callback in callbacks dict and run it

set_complect_pct(*val*)
Set the complete_pct to a new value

Parametersval : int/float

float value representing the new percentage to consider self.obj complete

setup_logging()
Setup loggers for this object

stop()

class pytan.pollers.**SSEPoller**(*handler*, *export_id*, ***kwargs*)
Bases: *pytan.pollers.QuestionPoller*

A class to poll the progress of a Server Side Export.

The primary function of this class is to poll for status of server side exports.

Parametershandler : *pytan.handler.Handler*

PyTan handler to use for GetResultInfo calls

export_id : str

•ID of server side export

polling_secs : int, optional

•default: 2

•Number of seconds to wait in between status check loops

timeout_secs : int, optional

•default: 600

•timeout in seconds for waiting for status completion, 0 does not time out

POLLING_SECS_DEFAULT = 2
default value for self.polling_secs

STR_ATTRS = ['export_id', 'polling_secs', 'timeout_secs', 'sse_status']
Class attributes to include in __str__ output

TIMEOUT_SECS_DEFAULT = 600
default value for self.timeout_secs

_post_init(***kwargs*)
Post init class setup

export_id = None
The export_id for this poller

get_sse_data(***kwargs*)
Function to get the data of a server side export

Constructs a URL via: export/${export_id}.gz and performs an authenticated HTTP get

get_sse_status(***kwargs*)
Function to get the status of a server side export

Constructs a URL via: export/${export_id}.status and performs an authenticated HTTP get

run(***kwargs*)
Poll for server side export status

sse_status_has_completed_loop(***kwargs*)
Method to poll the status file for a server side export until it contains 'Completed'

1.2.4 pytan.constants

PyTan Constants

This contains a number of constants that drive PyTan.

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': {'xml': [], 'json': [], 'csv': [{'valid_list_types': ['str', 'unicode'], 'key': 'h**

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, 'help': 'Filter for less than VALUE', 'human': ['<', '**

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, {'stats': 'DEBUG'}, 'Sets all loggers to only output at WARNING or above exc**

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
- str, description of this loglevel

pytan.constants.**OPTION_MAPS = [{'destination': 'filter', 'help': 'Make the filter do a case insensitive match', 'attrs': {'ig**

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'}, 'parsed': {'hand**
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: 'IPA**
Maps a Result type from the Tanium SOAP API from an int to a string

pytan.constants.**SSE_CRASH_MAP = ['6.5.314.4300']**
Mapping of versions to watch out for crashes/handle bugs for server side export

pytan.constants.**SSE_FORMAT_MAP = [('csv', '0', 0), ('xml', '1', 1), ('xml_obj', '1', 1), ('cef', '2', 2)]**
Mapping of human friendly strings to API integers for server side export

pytan.constants.**SSE_RESTRICT_MAP = {1: ['6.5.314.4300'], 2: ['6.5.314.4300']}**
Mapping of API integers for server side export format to version support

pytan.constants.**TIME_FORMAT = '%Y-%m-%dT%H:%M:%S'**
Tanium's format for date time strings

1.2.5 pytan.utils

Collection of classes and methods used throughout *pytan*

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*)

pytan.utils.**apply_options_obj**(*options*, *obj*, *dest*)
Updates an object with options

Parametersoptions : 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*)

Returnsobj : *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

Parametersq_filter_defs : list of dict

•List of dict that are question filter definitions

q_option_defs : dict

•dict of question filter options

Returnsgroup_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

Parametersselectlist_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

Returnsadd_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

Parametersproperties : 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

Returnsmetadatalist_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

Parameterskey : 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

Returnsparam_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

Parametersobj : *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 *pytan.exceptions.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 *pytan.exceptions.HandlerError*

•True: If user did not supply a value for a given parameter, do not add the parameter to the ParameterList object

Returnsparam_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

Parameterssensor_defs : list of dict

•List of dict that are sensor definitions

Returnsselect_objlist : *taniumpy.object_types.select_list.SelectList*

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

pytan.utils.**calc_percent**(*percent*, *whole*)
Utility method for getting percentage of whole

Parameterspercent: int, float

whole: int, float

Returnsint : the percentage of whole

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

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

Parametersd : 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.**check_for_help**(*kwargs*)
Utility method to check for any help arguments and raise a PytanHelp exception with the appropriate help

Parameterskwargs : dict

•dict of keyword args

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

Parametersdef_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 *pytan.exceptions.DefinitionParserError* if not

pytan.utils.**clean_kwargs**(*kwargs*, *keys=None*)

Removes each key from kwargs dict if found

Parameterskwargs : dict

•dict of keyword args

keys : list of str, optional

•default: ['obj', 'pytan_help', 'objtype']

•list of strs of keys to remove from kwargs

Returnsclean_kwargs : dict

•the new dict of kwargs with keys removed

pytan.utils.**copy_obj**(*obj*, *skip_attrs=None*)

Returns a new class of obj with with out any attributes in skip_attrs specified

Parametersobj : *taniumpy.object_types.base.BaseType*

•Object to copy

skip_attrs : list of str

•default: None

•list of attribute str's to skip copying over to new object, will default to [] if None

Returnsnew_obj : *taniumpy.object_types.base.BaseType*

•Copied object with attributes in skip_attrs skipped

pytan.utils.**copy_package_obj_for_action**(*obj*, *skip_attrs=None*)

Returns a new class of package obj with with out any attributes in skip_attrs specified

Parametersobj : *taniumpy.object_types.base.BaseType*

•Object to copy

skip_attrs : list of str

•default: None

•list of attribute str's to skip copying over to new object, default if None: ['id', 'deleted_flag', 'available_time', 'creation_time', 'modification_time', 'source_id']

Returnsnew_obj : *taniumpy.object_types.base.BaseType*

•Copied object with attributes in skip_attrs skipped

pytan.utils.**datetime_to_timestr**(*dt*)
Get a timestr for *dt*

Parametersdt : datetime.datetime

•datetime object

Returnstimestr: str

•the timestr for *dt* in taniums format

pytan.utils.**dehumanize_package**(*package*)
Turns a package str into a package definition

Parameterspackage : str

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

Returnspackage_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

Parametersquestion_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

Returnsquestion_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

Parametersquestion_options : str, list of str

•A str or list of str that describes question options

Returnsquestion_option_defs : list of dict

•list of dict parsed from *question_options*

pytan.utils.**dehumanize_sensors**(*sensors*, *key='sensors'*, *empty_ok=True*)
Turns a sensors str or list of str into a sensor definition

Parameterssensors : 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 *pytan.exceptions.HumanParserError* if it is empty

•True: *sensors* is allowed to be empty

Returnssensor_defs : list of dict

•list of dict parsed from *sensors*

pytan.utils.**derive_param_default**(*obj_param*)
Derive a parameter default

Parametersobj_param : dict

•parameter dict from TaniumPy object

Returnsdef_val : str

•default value derived from obj_param

pytan.utils.**empty_obj**(*taniumpy_object*)
Validate that a given TaniumPy object is not empty

Parameterstaniumpy_object : *taniumpy.object_types.base.BaseType*

•object to check if empty

Returnsbool

•True if *taniumpy_object* is considered empty, False otherwise

pytan.utils.**eval_timing**(*c*)
Yet another method to time things – c will be evaluated and timing information will be printed out

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

Parameterss : str

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

Returnss : 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*

Parameterss : str

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

Returnss : 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*

Parameterss : str

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

Returnss : 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*

Parameterss : 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

Returnss : str

•str *s* without the parsed_selector included

parsed_selector : str

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

pytan.utils.**func_timing**(*f*)
Decorator to add timing information around a function

pytan.utils.**get_all_loggers**()
Gets all loggers currently known to pythons logging system that exist in *pytan.constants.LOG_LEVEL_MAPS*

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

Parametersd : 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

Returnslist_len : int

•sum of lists in *d* that match keys

pytan.utils.**get_filter_obj**(*sensor_def*)
Creates a Filter object from sensor_def

Parameterssensor_def : dict

•dict containing sensor definition

Returnsfilter_obj : *taniumpy.object_types.filter.Filter*

•Filter object created from *sensor_def*

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

Parameterskey : 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

Returnsval : int

value from key, or default if supplied

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

Returnsstr :

str of current time return from *human_time()*

pytan.utils.**get_obj_map**(*objtype*)
Gets an object map for *objtype*

Parametersobjtype : str

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

Returnsobj_map : dict

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

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

Returnsparams : dict

•JSON loaded dict of parameters from *obj*

pytan.utils.**get_percentage**(*part*, *whole*)
Utility method for getting percentage of part out of whole

Parameterspart: int, float

whole: int, float

Returnsint : the percentage of part out of whole

pytan.utils.**get_q_obj_map**(*qtype*)
Gets an object map for *qtype*

Parametersqtype : str

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

Returnsobj_map : dict

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

pytan.utils.**get_taniumpy_obj**(*obj_map*)
Gets a taniumpy object from *obj_map*

Parametersobj_map : str

•str of taniumpy object to fetch

Returnsobj : *taniumpy.object_types.base.BaseType*

•matching taniumpy object for *obj_map*

pytan.utils.**human_time**(*t*, *tformat='%Y_%m_%d-%H_%M_%S-%Z'*)
Get time in human friendly format

Parameterst : 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

Returnsstr :

•*t* converted to str

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

pytan.utils.**jsonify**(*v*, *indent=2*, *sort_keys=True*)
Turns python object *v* into a pretty printed JSON string

Parametersv : 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

Returnsstr :

•JSON pretty printed string

pytan.utils.**load_param_json_file**(*parameters_json_file*)
Opens a json file and sanity checks it for use as a parameters element for a taniumpy object

Parametersparameters_json_file : str

•path to JSON file that describes an API object

Returnsobj

•contents of parameters_json_file de-serialized

pytan.utils.**load_taniumpy_from_json**(*json_file*)
Opens a json file and parses it into an taniumpy object

Parametersjson_file : str

•path to JSON file that describes an API object

Returnsobj : *taniumpy.object_types.base.BaseType*

•TaniumPy object converted from json file

pytan.utils.**log_session_communication**(*h*)
Uses *xml_pretty()* to pretty print the last request and response bodies from the session object in h to the logging system

Parametersh : Handler object

•Handler object with session object containing last request and response body

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

Parametersfilter_str : str

•filter_str str that should be validated

Returnsfilter_attrs : dict

•dict containing mapped filter attributes for SOAP API

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

Parametersopt : str

•option str that should be validated

dest : list of str

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

Returnsopt_attrs : dict

•dict containing mapped option attributes for SOAP API

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

Parametersoptions : list of str

•list of str that should be validated

dest : list of str

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

Returnsmapped_options : dict

•dict of all mapped_options

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

Parametersdefname : 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

Returnsnew_defs : list of dict

- parsed and validated defs

pytan.utils.**parse_versioning**(*server_version*)
Parses server_version into a dictionary

Parametersserver_version : str

- str of server version

Returnsdict

- dict of parsed tanium server version containing keys: major, minor, revision, and build

pytan.utils.**plugin_zip**(*p*)
Maps columns to values for each row in a plugins sql_response and returns a list of dicts

Parametersp : *taniumpy.object_types.plugin.Plugin*

- plugin object

Returnsdict

- the columns and result_rows of the sql_response in Plugin object zipped up into a dictionary

pytan.utils.**port_check**(*address*, *port*, *timeout=5*)
Check if *address*:*port* can be reached within *timeout*

Parametersaddress : 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

Returnssocket or False :

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

pytan.utils.**print_log_levels**()
Prints info about each loglevel from *pytan.constants.LOG_LEVEL_MAPS*

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

Parametersname : str

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

pytan.utils.**seconds_from_now**(*secs=0*, *tz='utc'*)
Get time in Tanium SOAP API format *secs* from now

Parameterssecs : 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

Returnsstr :

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

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*

Parametersloglevel : 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**(*gmt_tz=True*)
Creates a console logging handler using *SplitStreamHandler*

pytan.utils.**shrink_obj**(*obj*, *attrs=None*)
Returns a new class of obj with only id/name/hash defined

Parametersobj : *taniumpy.object_types.base.BaseType*

- Object to shrink

attrs : list of str

- default: None
- list of attribute str's to copy over to new object, will default to ['name', 'id', 'hash'] if None

Returnsnew_obj : *taniumpy.object_types.base.BaseType*

- Shrunken object

pytan.utils.**spew**(*t*)
Prints a string based on DEBUG_OUTPUT bool

Parameterst : str

- string to debug print

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

Parametershost : str

- hostname/ip address to check *port* on

port : int

- port to check on *host*

Raisespytan.exceptions.HandlerError : *pytan.exceptions.HandlerError*

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

pytan.utils.**timestr_to_datetime**(*timestr*)
Get a datetime.datetime object for *timestr*

Parameterstimestr : str

- date & time in taniums format

Returnsdatetime.datetime

- the datetime object for the timestr

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

Parameterspackage_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

Parametersq_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

Parameterssensor_defs : list of dict

•list of sensor definitions

pytan.utils.**xml_pretty**(*x*, *pretty=True*, *indent=' '*, ***kwargs*)
Uses *xmltodict* to pretty print an XML str *x*

Parametersx : str

•XML string to pretty print

Returnsstr :

•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*

Parametersx : str

•XML string to pretty print

Returnsstr :

•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*

Parametersx : str

•XML string to pretty print

Returnsstr :

•The pretty printed string of ResultXML in *x*

1.2.6 pytan.binsupport

Collection of classes and methods used throughout *pytan* for command line support

class pytan.binsupport.**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.binsupport.**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*)

class pytan.binsupport.**HistoryConsole**(*locals=None*, *filename='<console>'*, *histfile='/Users/jolsen/.console-history'*)
Bases: code.InteractiveConsole

Class that provides an interactive python console with full auto complete, history, and history file support.

Examples

```
>>> console = pytan.binsupport.HistoryConsole()
```

init_history(*histfile*)

static save_history(*histfile*)

pytan.binsupport.**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.binsupport.**add_file_log**(*logfile*, *debug=False*)
Utility to add a log file from python's logging module

pytan.binsupport.**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.binsupport.**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.binsupport.**csvdictwriter**(*rows_list*, ***kwargs*)
returns the rows_list (list of dicts) as a CSV string

pytan.binsupport.**debug_list**(*debuglist*)
Utility function to print the variables for a list of objects

pytan.binsupport.**debug_obj**(*debugobj*)
Utility function to print the variables for an object

pytan.binsupport.**filter_filename**(*filename*)
Utility to filter a string into a valid filename

pytan.binsupport.**filter_sensors**(*sensors*, *filter_platforms=[]*, *filter_categories=[]*)
Utility to filter a list of sensors for specific platforms and/or categories

pytan.binsupport.**filter_sourced_sensors**(*sensors*)
Utility to filter out all sensors that have a source_id specified (i.e. they are temp sensors created by the API)

pytan.binsupport.**get_all_headers**(*rows_list*)
Utility to get all the keys for a list of dicts

pytan.binsupport.**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.binsupport.**input_prompts**(*args*)
Utility function to prompt for username, password, and host if empty

pytan.binsupport.**introspect**(*obj*, *depth=0*)
Utility function to dump all info about an object

pytan.binsupport.**parse_sensor_platforms**(*sensor*)
Utility to create a list of platforms for a given sensor

pytan.binsupport.**print_obj**(*d*, *indent=0*)
Pretty print a dictionary

pytan.binsupport.**process_ask_manual_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for ask manual

Parameters
parser : argparse.ArgParse
•ArgParse object used to parse *all_args*
handler : *pytan.handler.Handler*
•Instance of Handler created from command line args
args : args object
•args parsed from *parser*
Returnsresponse
•response from *pytan.handler.Handler.ask_manual()*

pytan.binsupport.**process_ask_parsed_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for ask parsed

Parameters
parser : argparse.ArgParse
•ArgParse object used to parse *all_args*
handler : *pytan.handler.Handler*
•Instance of Handler created from command line args
args : args object
•args parsed from *parser*
Returnsresponse

•response from *pytan.handler.Handler.ask_parsed()*

pytan.binsupport.**process_ask_saved_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for ask saved

Parameters
parser : argparse.ArgParse
•ArgParse object used to parse *all_args*
handler : *pytan.handler.Handler*
•Instance of Handler created from command line args
args : args object
•args parsed from *parser*
Returnsresponse
•response from *pytan.handler.Handler.ask_saved()*

pytan.binsupport.**process_create_group_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for create group object

Parameters
parser : argparse.ArgParse
•ArgParse object used to parse *all_args*
handler : *pytan.handler.Handler*
•Instance of Handler created from command line args
args : args object
•args parsed from *parser*
Returns
response : *taniumpy.object_types.base.BaseType*
•response from *pytan.handler.Handler.create_group()*

pytan.binsupport.**process_create_json_object_args**(*parser*, *handler*, *obj*, *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
args : args object
•args parsed from *parser*
Returns
response : *taniumpy.object_types.base.BaseType*
•response from *pytan.handler.Handler.create_from_json()*

pytan.binsupport.**process_create_package_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for create package object

Parameters
parser : argparse.ArgParse
•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

Returnsresponse : *taniumpy.object_types.base.BaseType*

•response from *pytan.handler.Handler.create_package()*

pytan.binsupport.**process_create_sensor_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for create sensor object

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

Returnsresponse : *taniumpy.object_types.base.BaseType*

•response from *pytan.handler.Handler.create_sensor()*

pytan.binsupport.**process_create_user_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for create user object

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

Returnsresponse : *taniumpy.object_types.base.BaseType*

•response from *pytan.handler.Handler.create_user()*

pytan.binsupport.**process_create_whitelisted_url_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for create group object

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

Returnsresponse : *taniumpy.object_types.base.BaseType*

•response from *pytan.handler.Handler.create_group()*

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

Parametersparser : 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

args : args object

•args parsed from *parser*

Returnsresponse : *taniumpy.object_types.base.BaseType*

•response from *pytan.handler.Handler.delete()*

pytan.binsupport.**process_deploy_action_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for deploy action

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

Returnsresponse

•response from *pytan.handler.Handler.deploy_action()*

pytan.binsupport.**process_get_object_args**(*parser*, *handler*, *obj*, *args*, *report=True*)
Process command line args supplied by user for get object

Parametersparser : 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

args : args object

•args parsed from *parser*

Returnsresponse : *taniumpy.object_types.base.BaseType*

•response from *pytan.handler.Handler.get()*

pytan.binsupport.**process_get_results_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for getting results

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args

•args object from parsing *parser*

Returnsreport_path, report_contents : tuple

•results from *pytan.handler.Handler.export_to_report_file()* on the return of *pytan.handler.Handler.get_result_data()*

pytan.binsupport.**process_handler_args**(*parser*, *args*)
Process command line args supplied by user for handler

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

args : args

•args parsed from *parser*

Returnsh : *pytan.handler.Handler*

•Handler object

pytan.binsupport.**process_print_sensors_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for printing sensors

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

pytan.binsupport.**process_print_server_info_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for printing server info

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

pytan.binsupport.**process_pytan_shell_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for a python shell

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

pytan.binsupport.**process_stop_action_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for getting results

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args

•args object from parsing *parser*

Returnsreport_path, report_contents : tuple

•results from *pytan.handler.Handler.export_to_report_file()* on the return of *pytan.handler.Handler.get_result_data()*

pytan.binsupport.**process_tsat_args**(*parser*, *handler*, *args*)
Process command line args supplied by user for tsat

Parametersparser : argparse.ArgParse

•ArgParse object used to parse *all_args*

handler : *pytan.handler.Handler*

•Instance of Handler created from command line args

args : args object

•args parsed from *parser*

pytan.binsupport.**remove_file_log**(*logfile*)
Utility to remove a log file from python's logging module

pytan.binsupport.**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.binsupport.**setup_ask_parsed_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 parsed questions.

pytan.binsupport.**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.binsupport.**setup_create_group_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 create a group.

pytan.binsupport.**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.binsupport.**setup_create_package_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 create a package.

pytan.binsupport.**setup_create_sensor_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 create a sensor.

pytan.binsupport.**setup_create_user_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 create a user.

pytan.binsupport.**setup_create_whitelisted_url_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 create a whitelisted_url.

pytan.binsupport.**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.binsupport.**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.binsupport.**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.binsupport.**setup_get_results_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.binsupport.**setup_parent_parser**(*doc*)
Method to setup the base pytan.utils.CustomArgParse class for command line scripts using pytan.utils.setup_parser() and return a parser object for adding arguments to

pytan.binsupport.**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

pytan.binsupport.**setup_print_sensors_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 print server info.

pytan.binsupport.**setup_print_server_info_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 print sensor info.

pytan.binsupport.**setup_pytan_shell_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 create a python shell.

pytan.binsupport.**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.binsupport.**setup_tsat_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 objects.

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

Parametersreqver : str

•string containing version number to check against Exception

RaisesVersionMismatchError : Exception

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

1.2.7 pytan.exceptions

Provides exceptions for the *pytan* module.

exception pytan.exceptions.**AuthorizationError**
Bases: exceptions.Exception

Exception thrown for authorization errors in *pytan.sessions*

exception pytan.exceptions.**BadResponseError**
Bases: exceptions.Exception

Exception thrown for BadResponse messages from Tanium in *pytan.sessions*

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

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

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

Exception thrown for errors in *pytan.handler*

exception pytan.exceptions.**HttpError**
Bases: exceptions.Exception

Exception thrown for HTTP errors in *pytan.sessions*

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

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

exception pytan.exceptions.**NotFoundError**
Bases: exceptions.Exception

Exception thrown for Not Found messages from Tanium in *pytan.handler*

exception pytan.exceptions.**PickerError**
Bases: exceptions.Exception

Exception thrown for picker errors in *pytan.handler*

exception pytan.exceptions.**PollingError**
Bases: exceptions.Exception

Exception thrown for errors in pytan.polling

exception pytan.exceptions.**PytanHelp**
Bases: exceptions.Exception

Exception thrown when printing out help

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

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

exception pytan.exceptions.**ServerParseError**
Bases: exceptions.Exception

Exception thrown for server parsing errors in *pytan.handler*

exception pytan.exceptions.**ServerSideExportError**
Bases: exceptions.Exception

Exception thrown for server side export errors in *pytan.handler*

exception pytan.exceptions.**TimeoutException**
Bases: exceptions.Exception

Exception thrown for timeout errors in pytan.polling

exception pytan.exceptions.**UnsupportedVersionError**
Bases: exceptions.Exception

Exception thrown for version checks in *pytan.handler*

exception pytan.exceptions.**VersionMismatchError**
Bases: exceptions.Exception

Exception thrown for version_check in *pytan.utils*

exception pytan.exceptions.**VersionParseError**
Bases: exceptions.Exception

Exception thrown for server version parsing errors in *pytan.handler*

1.2.8 pytan.xml_clean

This is a regex based XML cleaner that will replace unsupported characters

pytan.xml_clean.**DEFAULT_REPLACEMENT = u'\ufffd'**
The default character to use when replacing characters

pytan.xml_clean.**INVALID_UNICODE_RAW_RE = u'[^\t\n\r -\ud7ff\ue000-\ufffd]'**
The raw regex string to use when replacing invalid characters

pytan.xml_clean.**INVALID_UNICODE_RE = <_sre.SRE_Pattern object>**
The regex object to use when replacing invalid characters

pytan.xml_clean.**RESTRICTED_UNICODE_RAW_RE = u'[\x7f-\x84\x86-\x9f\ufdd0-\ufdef]'**
The raw regex string to use when replacing restricted characters

pytan.xml_clean.**RESTRICTED_UNICODE_RE = <_sre.SRE_Pattern object>**
The regex object to use when replacing restricted characters

pytan.xml_clean.**XML_1_0_RESTRICTED_HEX = [[127, 132], [134, 159], [64976, 65007]]**

Restricted/discouraged Unicode characters for XML documents:[#x7F-#x84], [#x86-#x9F], [#xFDD0-#xFDEF], [#x1FFFE-#x1FFFF], [#x2FFFE-#x2FFFF], [#x3FFFE-#x3FFFF], [#x4FFFE-#x4FFFF], [#x5FFFE-#x5FFFF], [#x6FFFE-#x6FFFF], [#x7FFFE-#x7FFFF], [#x8FFFE-#x8FFFF], [#x9FFFE-#x9FFFF], [#xAFFFE-#xAFFFF], [#xBFFFE-#xBFFFF], [#xCFFFE-#xCFFFF], [#xDFFFE-#xDFFFF], [#xEFFFE-#xEFFFF], [#xFFFFE-#xFFFFF], [#x10FFFE-#x10FFFF]

Source: http://www.w3.org/TR/REC-xml/#NT-Char

pytan.xml_clean.**XML_1_0_VALID_HEX = [[9], [10], [13], [32, 55295], [57344, 65533]]**

Valid Unicode characters for XML documents:(any Unicode character, excluding the surrogate blocks, FFFE, and FFFF) #x9, #xA, #xD, [#x20-#xD7FF], [#xE000-#xFFFD], [#x10000-#x10FFFF]

Source: http://www.w3.org/TR/REC-xml/#NT-Char

pytan.xml_clean.**replace_invalid_unicode**(*text*, *replacement=None*)
Replaces invalid unicode characters with *replacement*

Parameterstext : str

•str to clean

replacement : str, optional

•default: None

•if invalid characters found, they will be replaced with this

•if not supplied, will default to DEFAULT_REPLACEMENT

Returnsstr, cnt, RE : tuple

•str : the cleaned version of *text*

•cnt : the number of replacements that took place

•RE : the regex object that was used to do the replacements

pytan.xml_clean.**replace_restricted_unicode**(*text*, *replacement=None*)
Replaces restricted unicode characters with *replacement*

Parameterstext : str

•str to clean

replacement : str, optional

•default: None

•if restricted characters found, they will be replaced with this

•if not supplied, will default to DEFAULT_REPLACEMENT

Returnsstr, cnt, RE : tuple

•str : the cleaned version of *text*

•cnt : the number of replacements that took place

•RE : the regex object that was used to do the replacements

pytan.xml_clean.**xml_cleaner**(*s*, *encoding='utf-8'*, *clean_restricted=True*, *log_clean_messages=True*, *log_bad_characters=False*, *replacement=None*, ***kwargs*)

Removes invalid /restricted characters per XML 1.0 spec

Parameterss : str

•str to clean

encoding : str, optional

•default: 'utf-8'

•encoding of *s*

clean_restricted : bool, optional

•default: True

•remove restricted characters from *s* or not

log_clean_messages : bool, optional

•default: True

•log messages using python logging or not

log_bad_characters : bool, optional

•default: False

•log bad character matches or not

Returnsstr

•the cleaned version of *s*

1.3 PyTan Tests

1.3.1 Valid Server Functional Tests

This contains valid 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.

These tests all use *ddt*, a package that provides for data driven tests via JSON files.

class test_pytan_valid_server_tests.**ValidServerTests**(*methodName='runTest'*)

Bases: unittest.case.TestCase

classmethod setUpClass()

setup_test()

classmethod tearDownClass()

test_invalid_create_object_1_invalid_create_sensor()

test_invalid_create_object_from_json_1_invalid_create_saved_action_from_json()

test_invalid_create_object_from_json_2_invalid_create_client_from_json()

test_invalid_create_object_from_json_3_invalid_create_userrole_from_json()

test_invalid_create_object_from_json_4_invalid_create_setting_from_json()

test_invalid_deploy_action_1_invalid_deploy_action_run_false()

test_invalid_deploy_action_2_invalid_deploy_action_package_help()

test_invalid_deploy_action_3_invalid_deploy_action_package()

test_invalid_deploy_action_4_invalid_deploy_action_options_help()

test_invalid_deploy_action_5_invalid_deploy_action_empty_package()

test_invalid_deploy_action_6_invalid_deploy_action_filters_help()

test_invalid_deploy_action_7_invalid_deploy_action_missing_parameters()

test_invalid_export_basetype_1_invalid_export_basetype_csv_bad_explode_type()

test_invalid_export_basetype_2_invalid_export_basetype_csv_bad_sort_sub_type()

test_invalid_export_basetype_3_invalid_export_basetype_csv_bad_sort_type()

test_invalid_export_basetype_4_invalid_export_basetype_xml_bad_minimal_type()

test_invalid_export_basetype_5_invalid_export_basetype_json_bad_include_type()

test_invalid_export_basetype_6_invalid_export_basetype_json_bad_explode_type()

test_invalid_export_basetype_7_invalid_export_basetype_bad_format()

test_invalid_export_resultset_1_invalid_export_resultset_csv_bad_sort_sub_type()

test_invalid_export_resultset_2_invalid_export_resultset_csv_bad_sort_type()

test_invalid_export_resultset_3_invalid_export_resultset_csv_bad_expand_type()

test_invalid_export_resultset_4_invalid_export_resultset_csv_bad_sensors_sub_type()

test_invalid_export_resultset_5_invalid_export_resultset_bad_format()

test_invalid_get_object_1_invalid_get_action_single_by_name()

test_invalid_get_object_2_invalid_get_question_by_name()

test_invalid_question_1_invalid_ask_manual_question_sensor_help()

test_invalid_question_2_invalid_ask_manual_question_bad_filter()

test_invalid_question_3_invalid_ask_manual_question_filter_help()

test_invalid_question_4_invalid_ask_manual_question_bad_option()

test_invalid_question_5_invalid_ask_manual_question_missing_parameter_split()

test_invalid_question_6_invalid_ask_manual_question_option_help()

test_invalid_question_7_invalid_ask_manual_question_too_many_parameter_blocks()

test_invalid_question_8_invalid_ask_manual_question_bad_sensorname()

test_valid_create_object_1_create_user()

test_valid_create_object_2_create_package()

test_valid_create_object_3_create_group()

test_valid_create_object_4_create_whitelisted_url()

test_valid_create_object_from_json_1_create_package_from_json()

test_valid_create_object_from_json_2_create_user_from_json()

test_valid_create_object_from_json_3_create_saved_question_from_json()

test_valid_create_object_from_json_4_create_action_from_json()

test_valid_create_object_from_json_5_create_sensor_from_json()

test_valid_create_object_from_json_6_create_question_from_json()

test_valid_create_object_from_json_7_create_whitelisted_url_from_json()

test_valid_create_object_from_json_8_create_group_from_json()

test_valid_deploy_action_1_deploy_action_simple_against_windows_computers()

test_valid_deploy_action_2_deploy_action_simple_without_results()

test_valid_deploy_action_3_deploy_action_with_params_against_windows_computers()

test_valid_deploy_action_4_deploy_action_simple()

test_valid_export_basetype_10_export_basetype_xml_default_options()

test_valid_export_basetype_11_export_basetype_csv_with_explode_true()

test_valid_export_basetype_12_export_basetype_json_explode_false()

test_valid_export_basetype_13_export_basetype_json_type_false()

test_valid_export_basetype_14_export_basetype_json_default_options()

test_valid_export_basetype_1_export_basetype_csv_with_sort_list()

test_valid_export_basetype_2_export_basetype_csv_with_explode_false()

test_valid_export_basetype_3_export_basetype_json_type_true()

test_valid_export_basetype_4_export_basetype_xml_minimal_false()

test_valid_export_basetype_5_export_basetype_xml_minimal_true()

test_valid_export_basetype_6_export_basetype_csv_with_sort_empty_list()

test_valid_export_basetype_7_export_basetype_csv_default_options()

test_valid_export_basetype_8_export_basetype_json_explode_true()

test_valid_export_basetype_9_export_basetype_csv_with_sort_true()

test_valid_export_resultset_10_export_resultset_csv_default_options()

test_valid_export_resultset_11_export_resultset_csv_type_true()

test_valid_export_resultset_12_export_resultset_csv_all_options()

test_valid_export_resultset_13_export_resultset_csv_sort_false()

test_valid_export_resultset_1_export_resultset_json()

test_valid_export_resultset_2_export_resultset_csv_sensor_true()

test_valid_export_resultset_3_export_resultset_csv_type_false()

test_valid_export_resultset_4_export_resultset_csv_expand_false()

test_valid_export_resultset_5_export_resultset_csv_sort_empty()

test_valid_export_resultset_6_export_resultset_csv_sort_true()

test_valid_export_resultset_7_export_resultset_csv_sort_list()

test_valid_export_resultset_8_export_resultset_csv_sensor_false()

test_valid_export_resultset_9_export_resultset_csv_expand_true()

test_valid_get_object_10_get_all_saved_questions()

test_valid_get_object_11_get_user_by_name()

test_valid_get_object_12_get_all_userroless()

test_valid_get_object_13_get_all_questions()

test_valid_get_object_14_get_sensor_by_id()

test_valid_get_object_15_get_all_groups()

test_valid_get_object_16_get_all_sensors()

test_valid_get_object_17_get_sensor_by_mixed()

test_valid_get_object_18_get_whitelisted_url_by_id()

test_valid_get_object_19_get_group_by_name()

test_valid_get_object_1_get_all_users()

test_valid_get_object_20_get_all_whitelisted_urls()

test_valid_get_object_21_get_sensor_by_hash()

test_valid_get_object_22_get_package_by_name()

test_valid_get_object_23_get_all_clients()

test_valid_get_object_24_get_sensor_by_names()

test_valid_get_object_25_get_all_packages()

test_valid_get_object_26_get_saved_question_by_name()

test_valid_get_object_27_get_all_actions()

test_valid_get_object_28_get_user_by_id()

test_valid_get_object_29_get_sensor_by_name()

test_valid_get_object_2_get_action_by_id()

test_valid_get_object_30_get_saved_action_by_name()

test_valid_get_object_3_get_question_by_id()

test_valid_get_object_4_get_saved_question_by_names()

test_valid_get_object_5_get_userrole_by_id()

test_valid_get_object_6_get_all_saved_actions()

test_valid_get_object_7_get_leader_clients()

test_valid_get_object_8_get_all_settings()

test_valid_get_object_9_get_setting_by_name()

test_valid_question_10_ask_manual_question_sensor_with_filter()

test_valid_question_11_ask_manual_question_multiple_sensors_identified_by_name()

test_valid_question_12_ask_manual_question_sensor_with_parameters_and_filter_and_optio

test_valid_question_13_ask_manual_question_sensor_with_filter_and_3_options()

test_valid_question_14_ask_manual_question_complex_query2()

test_valid_question_15_ask_manual_question_complex_query1()

test_valid_question_1_ask_manual_question_sensor_with_parameters_and_some_supplied_par

test_valid_question_2_ask_manual_question_multiple_sensors_with_parameters_and_some_su

test_valid_question_3_ask_manual_question_simple_multiple_sensors()

test_valid_question_4_ask_manual_question_sensor_without_parameters_and_supplied_param

test_valid_question_5_ask_manual_question_sensor_with_filter_and_2_options()

test_valid_question_6_ask_manual_question_sensor_with_parameters_and_filter()

test_valid_question_7__ask_manual_question_sensor_complex()

test_valid_question_8_ask_manual_question_sensor_with_parameters_and_no_supplied_param

test_valid_question_9_ask_manual_question_simple_single_sensor()

test_valid_saved_question_1_ask_saved_question_refresh_data()

test_valid_saved_question_2_ask_saved_question_by_name()

test_valid_saved_question_3_ask_saved_question_by_name_in_list()

test_pytan_valid_server_tests.**chew_csv**(*c*)

test_pytan_valid_server_tests.**spew**(*m*, *l=3*)

1.3.2 Invalid Server Functional Tests

This contains invalid 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.

These tests all use *ddt*, a package that provides for data driven tests via JSON files.

class test_pytan_invalid_server_tests.**InvalidServerTests**(*methodName='runTest'*)

Bases: unittest.case.TestCase

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()

test_pytan_invalid_server_tests.**spew**(*m*, *l=3*)

1.3.3 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

test_extract_filter_invalid()

test_extract_filter_nofilter()

test_extract_filter_valid()

test_extract_filter_valid_all()

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

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

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

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.**TestDeserializeBadXML**(*methodName='runTest'*)
Bases: unittest.case.TestCase

test_bad_chars_basetype_control()
This XML file has a number of control characters that are not valid in XML.

This test validates that pytan.xml_clean.xml_cleaner() will remove all the invalid and restricted characters, which should allow the body to be parsed properly.

test_bad_chars_resultset_latin1()
This XML file has some characters that are actually encoded as latin1 (as well as some restricted characters).

This test validates that pytan.xml_clean.xml_cleaner() will properly fall back to latin1 for decoding the docuemnt, as well as remove all the invalid and restricted characters, which should allow the body to be parsed properly.

test_bad_chars_resultset_surrogate()
This XML file has some characters that are unpaired surrogates in unicode. Surrogates (unpaired or otherwise) are not legal XML characters.

This test validates that pytan.xml_clean.xml_cleaner() will properly remove all the invalid and restricted characters, which should allow the body to be parsed properly.

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

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_load_param_file_invalid_file()

test_load_param_file_invalid_json()

test_load_param_file_valid()

test_load_taniumpy_file_invalid_file()

test_load_taniumpy_file_invalid_json()

test_version_higher()

test_version_lower()

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

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 added as of 1.0.4)

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

test_invalid1()

test_invalid2()

test_valid1()

test_valid2()

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

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

test_invalid1()

test_valid1()

test_valid2()

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

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

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

test_invalid1()

test_invalid2()

test_invalid3()

test_invalid4()

test_valid1()

test_valid2()

test_valid3()

test_valid4()

1.4 TaniumPy Package

A python package that handles the serialization/deserialization of XML SOAP requests/responses from Tanium to/from python objects.

1.4.1 Subpackages

taniumpy.object_types package

Submodules

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.permission_list module

class taniumpy.object_types.permission_list.**PermissionList**
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 ResultSet from a result_set SOAPElement

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_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.string_hint_list module

class taniumpy.object_types.string_hint_list.**StringHintList**
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_setting_list module

class taniumpy.object_types.system_setting_list.**SystemSettingList**
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_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*

1.5 Other Packages

PyTan relies on a number of python packages to function properly. All dependencies are bundled with PyTan in order to make it easier for the user to start using PyTan right away.

1.5.1 requests Package

PyTan uses requests for all HTTP requests in order to get automatic keep alive support, session tracking, and a host of other things. requests is an open source package maintained at: https://github.com/kennethreitz/requests

requests HTTP library

Requests is an HTTP library, written in Python, for human beings. Basic GET usage:

```
>>> import requests
>>> r = requests.get('https://www.python.org')
>>> r.status_code
200
>>> 'Python is a programming language' in r.content
True
```

... or POST:

```
>>> payload = dict(key1='value1', key2='value2')
>>> r = requests.post('http://httpbin.org/post', data=payload)
>>> print(r.text)
{
  ...
  "form": {
    "key2": "value2",
    "key1": "value1"
  },
  ...
}
```

The other HTTP methods are supported - see *requests.api*. Full documentation is at <http://python-requests.org>.

copyright

3. 2015 by Kenneth Reitz.

license Apache 2.0, see LICENSE for more details.

1.5.2 threaded_http Package

PyTan uses threaded_http to create a fake HTTP server on localhost for the invalid server functional tests (see: `pytan.test_pytan_invalid_server_tests`). threaded_http is developed and maintained by Tanium. Simple HTTP server for testing purposes

class `threaded_http.`**`CustomHTTPHandler`**(*request*, *client_address*, *server*)

Bases: `BaseHTTPServer.BaseHTTPRequestHandler`

`ENABLE_LOGGING` = **True**

`do_GET`()

`do_POST`()

`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.

`threaded_http.`**`threaded_http`**(*host='localhost'*, *port=4443*, *verbosity=2*)

establishes an HTTP server on host:port in a thread

1.5.3 xmltodict Package

PyTan uses xmltodict for pretty printing XML documents (see: *`pytan.utils.xml_pretty()`*). xmltodict is an open source package maintained at: https://github.com/martinblech/xmltodict 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.

1.5.4 ddt Package

PyTan uses ddt for creating automatically generating test cases from JSON files (see: `pytan.test_pytan_valid_server_tests`). ddt is an open source package maintained at: https://github.com/txels/ddt

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.

1.6 PyTan API Examples

Each of these sections contains examples that show Example Python code for using a PyTan method, along with the standard output and standard error from running each example

1.6.1 PyTan API Basic Handler Example

This is an example for how to instantiate a `pytan.Handler` object.

The username, password, host, and maybe port as well need to be provided on a per Tanium server basis.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
```

```
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)
```

1.6.2 PyTan API Valid Create Object Examples

All of the PyTan API examples for Valid Create Object

Create User

Create a user called API Test User

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.delete() method
delete_kwargs = {}
delete_kwargs["objtype"] = u'user'
delete_kwargs["name"] = u'API Test User'

# setup the arguments for the handler() class
kwargs = {}
kwargs["rolename"] = u'Administrator'
kwargs["name"] = u'API Test User'
kwargs["properties"] = [[u'property1', u'value1']]

# delete the object in case it already exists
```

```
# catch and print the exception error if it does not exist
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
try:
    handler.delete(**delete_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)

print "...CALLING: handler.create_user() with args: {}".format(kwargs)
response = handler.create_user(**kwargs)

print "...OUTPUT: Type of response: ", type(response)
print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'
print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out

# delete the object, we are done with it now
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
delete_response = handler.delete(**delete_kwargs)

print "...OUTPUT: print the delete response"
print delete_response
```

Create Package

Create a package called package49

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.delete() method
delete_kwargs = {}
delete_kwargs["objtype"] = u'package'
delete_kwargs["name"] = u'package49'

# setup the arguments for the handler() class
kwargs = {}
kwargs["expire_seconds"] = 1500
kwargs["display_name"] = u'package49 API test'
kwargs["name"] = u'package49'
kwargs["parameters_json_file"] = u'../doc/example_of_all_package_parameters.json'
kwargs["verify_expire_seconds"] = 3600
```

```
kwargs["command"] = u'package49 $1 $2 $3 $4 $5 $6 $7 $8'
kwargs["file_urls"] = [u'3600::testing.vbs||https://content.tanium.com/files/initialcontent/bundles/2
kwargs["verify_filter_options"] = [u'and']
kwargs["verify_filters"] = [u'Custom Tags, that contains:tag']
kwargs["command_timeout_seconds"] = 9999

# delete the object in case it already exists
# catch and print the exception error if it does not exist
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
try:
    handler.delete(**delete_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)

print "...CALLING: handler.create_package() with args: {}".format(kwargs)
response = handler.create_package(**kwargs)

print "...OUTPUT: Type of response: ", type(response)
print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'
print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out

# delete the object, we are done with it now
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
delete_response = handler.delete(**delete_kwargs)

print "...OUTPUT: print the delete response"
print delete_response
```

Create Group

Create a group called All Windows Computers API Test

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
```

```
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.delete() method
delete_kwargs = {}
delete_kwargs["objtype"] = u'group'
delete_kwargs["name"] = u'All Windows Computers API Test'

```

```
# setup the arguments for the handler() class
kwargs = {}
kwargs["groupname"] = u'All Windows Computers API Test'
kwargs["filters"] = [u'Operating System, that contains:Windows']
kwargs["filter_options"] = [u'and']

# delete the object in case it already exists
# catch and print the exception error if it does not exist
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
try:
    handler.delete(**delete_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)

print "...CALLING: handler.create_group() with args: {}".format(kwargs)
response = handler.create_group(**kwargs)

print "...OUTPUT: Type of response: ", type(response)
print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'
print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out

# delete the object, we are done with it now
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
delete_response = handler.delete(**delete_kwargs)

print "...OUTPUT: print the delete response"
print delete_response
```

Create Whitelisted Url

Create a whitelisted url

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
```

```
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.delete() method
delete_kwargs = {}
delete_kwargs["objtype"] = u'whitelisted_url'
delete_kwargs["url_regex"] = u'regex:http://test.com/.*API_Test.*URL'

```

```
# setup the arguments for the handler() class
kwargs = {}
kwargs["url"] = u'http://test.com/.*API_Test.*URL'
kwargs["regex"] = True
kwargs["properties"] = [[u'property1', u'value1']]
kwargs["download_seconds"] = 3600

# delete the object in case it already exists
# catch and print the exception error if it does not exist
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
try:
    handler.delete(**delete_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)

print "...CALLING: handler.create_whitelisted_url() with args: {}".format(kwargs)
response = handler.create_whitelisted_url(**kwargs)

print "...OUTPUT: Type of response: ", type(response)
print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'
print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out

# delete the object, we are done with it now
print "...CALLING: handler.delete() with args: {}".format(delete_kwargs)
delete_response = handler.delete(**delete_kwargs)

print "...OUTPUT: print the delete response"
print delete_response
```

1.6.3 PyTan API Valid Create Object From JSON Examples

All of the PyTan API examples for Valid Create Object From JSON

Create Package From JSON

Export a package object to a JSON file, adding ' API TEST' to the name of the package before exporting the JSON file and deleting any pre-existing package with the same (new) name, then create a new package object from the exported JSON file

- STDOUT from Example Python Code

- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
```

```
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'package'
get_kwargs["id"] = 31

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# set the attribute name and value we want to add to the original objects
# this is necessarry to avoid name conflicts when adding the new object
attr_name = u'name'
attr_value = u' API TEST'

# modify the orig_objs to add attr_value to attr_name
for x in orig_objs:
    new_attr = getattr(x, attr_name)
    new_attr += attr_value
    setattr(x, attr_name, new_attr)

    # delete the object in case it already exists
    del_kwargs = {}
    del_kwargs[attr_name] = new_attr
    del_kwargs['objtype'] = u'package'

    print "...CALLING: handler.delete() with args: {}".format(del_kwargs)
    try:
        handler.delete(**del_kwargs)
    except Exception as e:
        print "...EXCEPTION: {}".format(e)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'package'
create_kwargs['json_file'] = json_file

print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
response = handler.create_from_json(**create_kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
```

```
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Create User From JSON

Export a user object to a JSON file, adding ' API TEST' to the name of the user before exporting the JSON file and deleting any pre-existing user with the same (new) name, then create a new user object from the exported JSON file

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

```

```
# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'user'
get_kwargs["id"] = 1

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# set the attribute name and value we want to add to the original objects
# this is necessarry to avoid name conflicts when adding the new object
attr_name = u'name'
attr_value = u' API TEST'

# modify the orig_objs to add attr_value to attr_name
for x in orig_objs:
    new_attr = getattr(x, attr_name)
    new_attr += attr_value
    setattr(x, attr_name, new_attr)

    # delete the object in case it already exists
    del_kwargs = {}
    del_kwargs[attr_name] = new_attr
    del_kwargs['objtype'] = u'user'

    print "...CALLING: handler.delete() with args: {}".format(del_kwargs)
    try:
        handler.delete(**del_kwargs)
    except Exception as e:
        print "...EXCEPTION: {}".format(e)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
```

```
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'user'
create_kwargs['json_file'] = json_file

print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
response = handler.create_from_json(**create_kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Create Saved Question From JSON

Export a saved question object to a JSON file, adding ' API TEST' to the name of the saved question before exporting the JSON file and deleting any pre-existing saved question with the same (new) name, then create a new saved question object from the exported JSON file

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

```

```
# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'saved_question'
get_kwargs["id"] = 1

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# set the attribute name and value we want to add to the original objects
# this is necessarry to avoid name conflicts when adding the new object
```

```
attr_name = u'name'
attr_value = u' API TEST'

# modify the orig_objs to add attr_value to attr_name
for x in orig_objs:
    new_attr = getattr(x, attr_name)
    new_attr += attr_value
    setattr(x, attr_name, new_attr)

    # delete the object in case it already exists
    del_kwargs = {}
    del_kwargs[attr_name] = new_attr
    del_kwargs['objtype'] = u'saved_question'

    print "...CALLING: handler.delete() with args: {}".format(del_kwargs)
    try:
        handler.delete(**del_kwargs)
    except Exception as e:
        print "...EXCEPTION: {}".format(e)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'saved_question'
create_kwargs['json_file'] = json_file

print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
response = handler.create_from_json(**create_kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Create Action From JSON

Export an action object to a JSON file, then create a new action object from the exported JSON file. Actions can not be deleted, so do not delete it. This will, in effect, 're-deploy' an action.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True
```

```
50
51  # instantiate a handler using all of the arguments in the handler_args dictionary
52  print "...CALLING: pytan.handler() with args: {}".format(handler_args)
53  handler = pytan.Handler(**handler_args)
54
55  # print out the handler string
56  print "...OUTPUT: handler string: {}".format(handler)
57
58  # setup the arguments for the handler.get() method
59  get_kwargs = {}
60  get_kwargs["objtype"] = u'action'
61  get_kwargs["id"] = 1
62
63  # get objects to use as an export to JSON file
64  print "...CALLING: handler.get() with args: {}".format(get_kwargs)
65  orig_objs = handler.get(**get_kwargs)
66
67  # export orig_objs to a json file
68  export_kwargs = {}
69  export_kwargs['obj'] = orig_objs
70  export_kwargs['export_format'] = 'json'
71  export_kwargs['report_dir'] = tempfile.gettempdir()
72
73  print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
74  json_file, results = handler.export_to_report_file(**export_kwargs)
75
76  # create the object from the exported JSON file
77  create_kwargs = {}
78  create_kwargs['objtype'] = u'action'
79  create_kwargs['json_file'] = json_file
80
81  print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
82  response = handler.create_from_json(**create_kwargs)
83
84  print "...OUTPUT: Type of response: ", type(response)
85
86  print "...OUTPUT: print of response:"
87  print response
88
89  # call the export_obj() method to convert response to JSON and store it in out
90  export_kwargs = {}
91  export_kwargs['obj'] = response
92  export_kwargs['export_format'] = 'json'
93
94  print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
95  out = handler.export_obj(**export_kwargs)
96
97  # trim the output if it is more than 15 lines long
98  if len(out.splitlines()) > 15:
99      out = out.splitlines()[0:15]
100     out.append('..trimmed for brevity..')
101     out = '\n'.join(out)
102
103 print "...OUTPUT: print the objects returned in JSON format:"
104 print out
```

Create Sensor From JSON

Export a sensor object to a JSON file, adding ' API TEST' to the name of the sensor before exporting the JSON file and deleting any pre-existing sensor with the same (new) name, then create a new sensor object from the exported JSON file

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
```

```
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'sensor'
get_kwargs["id"] = 381

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# set the attribute name and value we want to add to the original objects
# this is necessarry to avoid name conflicts when adding the new object
attr_name = u'name'
attr_value = u' API TEST'

# modify the orig_objs to add attr_value to attr_name
for x in orig_objs:
    new_attr = getattr(x, attr_name)
    new_attr += attr_value
    setattr(x, attr_name, new_attr)

    # delete the object in case it already exists
    del_kwargs = {}
    del_kwargs[attr_name] = new_attr
    del_kwargs['objtype'] = u'sensor'

    print "...CALLING: handler.delete() with args: {}".format(del_kwargs)
    try:
        handler.delete(**del_kwargs)
    except Exception as e:
        print "...EXCEPTION: {}".format(e)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'sensor'
create_kwargs['json_file'] = json_file

print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
response = handler.create_from_json(**create_kwargs)

print "...OUTPUT: Type of response: ", type(response)
```

```

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Create Question From JSON

Export a question object to a JSON file, then create a new question object from the exported JSON file. Questions can not be deleted, so do not delete it. This will, in effect, 're-ask' a question.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]
```

```

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'question'
get_kwargs["id"] = 1

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'question'
create_kwargs['json_file'] = json_file

print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
response = handler.create_from_json(**create_kwargs)

print "...OUTPUT: Type of response: ", type(response)
```

```

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Create Whitelisted Url From JSON

Export a whitelisted url object to a JSON file, adding ' test1' to the url_regex of the whitelisted url before exporting the JSON file and deleting any pre-existing whitelisted url with the same (new) name, then create a new whitelisted url object from the exported JSON file

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
```

```
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'whitelisted_url'
get_kwargs["url_regex"] = u'test1'

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# set the attribute name and value we want to add to the original objects
# this is necessarry to avoid name conflicts when adding the new object
attr_name = u'url_regex'
attr_value = u' API TEST'

# modify the orig_objs to add attr_value to attr_name
for x in orig_objs:
    new_attr = getattr(x, attr_name)
    new_attr += attr_value
    setattr(x, attr_name, new_attr)

    # delete the object in case it already exists
    del_kwargs = {}
    del_kwargs[attr_name] = new_attr
    del_kwargs['objtype'] = u'whitelisted_url'

    print "...CALLING: handler.delete() with args: {}".format(del_kwargs)
```

```
    try:
        handler.delete(**del_kwargs)
    except Exception as e:
        print "...EXCEPTION: {}".format(e)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'whitelisted_url'
create_kwargs['json_file'] = json_file

print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
response = handler.create_from_json(**create_kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Create Group From JSON

Export a group object to a JSON file, adding ' API TEST' to the name of the group before exporting the JSON file and deleting any pre-existing group with the same (new) name, then create a new group object from the exported JSON file

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
```

```
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'group'
```

```
get_kwargs["name"] = u'All Computers'

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# set the attribute name and value we want to add to the original objects
# this is necessarry to avoid name conflicts when adding the new object
attr_name = u'name'
attr_value = u' API TEST'

# modify the orig_objs to add attr_value to attr_name
for x in orig_objs:
    new_attr = getattr(x, attr_name)
    new_attr += attr_value
    setattr(x, attr_name, new_attr)

    # delete the object in case it already exists
    del_kwargs = {}
    del_kwargs[attr_name] = new_attr
    del_kwargs['objtype'] = u'group'

    print "...CALLING: handler.delete() with args: {}".format(del_kwargs)
    try:
        handler.delete(**del_kwargs)
    except Exception as e:
        print "...EXCEPTION: {}".format(e)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'group'
create_kwargs['json_file'] = json_file

print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
response = handler.create_from_json(**create_kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

```

```
# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

1.6.4 PyTan API Valid Deploy Action Examples

All of the PyTan API examples for Valid Deploy Action

Deploy Action Simple

Deploy an action against all computers using human strings.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
```

```
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["run"] = True
kwargs["package"] = u'Distribute Tanium Standard Utilities'

print "...CALLING: handler.deploy_action with args: {}".format(kwargs)
response = handler.deploy_action(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Print of action object: "
print response['action_object']

# if results were returned (i.e. get_results=True was one of the kwargs passed in):
if response['action_results']:
    # call the export_obj() method to convert response to CSV and store it in out
    export_kwargs = {}
    export_kwargs['obj'] = response['action_results']
    export_kwargs['export_format'] = 'csv'
    print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
    out = handler.export_obj(**export_kwargs)

    # trim the output if it is more than 15 lines long
    if len(out.splitlines()) > 15:
        out = out.splitlines()[0:15]
        out.append('..trimmed for brevity..')
        out = '\n'.join(out)

    print "...OUTPUT: CSV Results of response: "
    print out
```

Deploy Action Simple Without Results

Deploy an action against all computers using human strings, but do not get the completed results of the job – return right away with the deploy action object.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True
```

```

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["get_results"] = False
kwargs["run"] = True
kwargs["package"] = u'Distribute Tanium Standard Utilities'

print "...CALLING: handler.deploy_action with args: {}".format(kwargs)
response = handler.deploy_action(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Print of action object: "
print response['action_object']

# if results were returned (i.e. get_results=True was one of the kwargs passed in):
if response['action_results']:
    # call the export_obj() method to convert response to CSV and store it in out
    export_kwargs = {}
    export_kwargs['obj'] = response['action_results']
    export_kwargs['export_format'] = 'csv'
    print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
    out = handler.export_obj(**export_kwargs)

    # trim the output if it is more than 15 lines long
    if len(out.splitlines()) > 15:
        out = out.splitlines()[0:15]
        out.append('..trimmed for brevity..')
        out = '\n'.join(out)

    print "...OUTPUT: CSV Results of response: "
    print out
```

Deploy Action Simple Against Windows Computers

Deploy an action against only windows computers using human strings. This requires passing in an action filter

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
```

```
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["run"] = True
kwargs["action_filters"] = u'Operating System, that contains:Windows'
kwargs["package"] = u'Distribute Tanium Standard Utilities'

```

```
print "...CALLING: handler.deploy_action with args: {}".format(kwargs)
response = handler.deploy_action(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Print of action object: "
print response['action_object']

# if results were returned (i.e. get_results=True was one of the kwargs passed in):
if response['action_results']:
    # call the export_obj() method to convert response to CSV and store it in out
    export_kwargs = {}
    export_kwargs['obj'] = response['action_results']
    export_kwargs['export_format'] = 'csv'
    print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
    out = handler.export_obj(**export_kwargs)

    # trim the output if it is more than 15 lines long
    if len(out.splitlines()) > 15:
        out = out.splitlines()[0:15]
        out.append('..trimmed for brevity..')
        out = '\n'.join(out)

    print "...OUTPUT: CSV Results of response: "
    print out
```

Deploy Action With Params Against Windows Computers

Deploy an action with parameters against only windows computers using human strings.

This will use the Package 'Custom Tagging - Add Tags' and supply two parameters. The second parameter will be ignored because the package in question only requires one parameter.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
```

```
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["run"] = True
kwargs["action_filters"] = u'Operating System, that contains:Windows'
kwargs["package"] = u'Custom Tagging - Add Tags{$1=tag_should_be_added,$2=tag_should_be_ignore}'

print "...CALLING: handler.deploy_action with args: {}".format(kwargs)
response = handler.deploy_action(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Print of action object: "
print response['action_object']

```

```
# if results were returned (i.e. get_results=True was one of the kwargs passed in):
if response['action_results']:
    # call the export_obj() method to convert response to CSV and store it in out
    export_kwargs = {}
    export_kwargs['obj'] = response['action_results']
    export_kwargs['export_format'] = 'csv'
    print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
    out = handler.export_obj(**export_kwargs)

    # trim the output if it is more than 15 lines long
    if len(out.splitlines()) > 15:
        out = out.splitlines()[0:15]
        out.append('..trimmed for brevity..')
        out = '\n'.join(out)

    print "...OUTPUT: CSV Results of response: "
    print out
```

1.6.5 PyTan API Valid Export Basetype Examples

All of the PyTan API examples for Valid Export Basetype

Export Basetype CSV Default Options

Export a BaseType from getting objects as CSV with the default options

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
```

```
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
```

```
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype JSON Type False

Export a BaseType from getting objects as JSON with false for include_type

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
```

```
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'
kwargs["include_type"] = False

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype JSON Explode False

Export a BaseType from getting objects as JSON with false for explode_json_string_values

- STDOUT from Example Python Code

- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
```

```
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'
kwargs["explode_json_string_values"] = False

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype JSON Explode True

Export a BaseType from getting objects as JSON with true for explode_json_string_values

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'
kwargs["explode_json_string_values"] = True

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
```

```
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype XML Default Options

Export a BaseType from getting objects as XML with the default options

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
```

```
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'xml'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
```

```
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype XML Minimal False

Export a BaseType from getting objects as XML with false for minimal

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
```

```
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'xml'
kwargs["minimal"] = False

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype XML Minimal True

Export a BaseType from getting objects as XML with true for minimal

- STDOUT from Example Python Code

- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
```

```
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'xml'
kwargs["minimal"] = True

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype CSV With Explode False

Export a BaseType from getting objects as CSV with false for explode_json_string_values

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["explode_json_string_values"] = False

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
```

```
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype CSV With Explode True

Export a BaseType from getting objects as CSV with true for explode_json_string_values

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
```

```
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["explode_json_string_values"] = True

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

```

```
# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype CSV With Sort Empty List

Export a BaseType from getting objects as CSV with an empty list for header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

```

```
# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = []

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype CSV With Sort True

Export a BaseType from getting objects as CSV with true for header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```python
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

```

```
# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = True

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype CSV With Sort List

Export a BaseType from getting objects as CSV with name and description for header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

```

```
# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = [u'name', u'description']

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
```

```
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype JSON Default Options

Export a BaseType from getting objects as JSON with the default options

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
```

```
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

```

```
# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export Basetype JSON Type True

Export a BaseType from getting objects as JSON with true for include_type

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

```

```
# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'
kwargs["include_type"] = True

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to export
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

1.6.6 PyTan API Valid Export ResultSet Examples

All of the PyTan API examples for Valid Export ResultSet

Export ResultSet CSV Default Options

Export a ResultSet from asking a question as CSV with the default options

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Expand False

Export a ResultSet from asking a question as CSV with false for expand_grouped_columns

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

```

```
# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["expand_grouped_columns"] = False

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
```

```
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Expand True

Export a ResultSet from asking a question as CSV with true for expand_grouped_columns

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
```

```
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["expand_grouped_columns"] = True

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
```

```
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV All Options

Export a ResultSet from asking a question as CSV with true for header_add_sensor, true for header_add_type, true for header_sort, and true for expand_grouped_columns

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
```

```
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["header_sort"] = True
kwargs["export_format"] = u'csv'
kwargs["header_add_type"] = True
kwargs["expand_grouped_columns"] = True
kwargs["header_add_sensor"] = True

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

```

```
print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet JSON

Export a ResultSet from asking a question as JSON with the default options

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False
```

```

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Sort Empty

Export a ResultSet from asking a question as CSV with an empty list for header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
```

```
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
```

```
kwargs["header_sort"] = []

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Sort True

Export a ResultSet from asking a question as CSV with true for header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
```

```
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = True

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)
```

```

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Sort False

Export a ResultSet from asking a question as CSV with false for header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

```

```
# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = False

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)
```

```
print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Sort List

Export a ResultSet from asking a question as CSV with Computer Name and IP Address for the header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
```

```
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = [u'Computer Name', u'IP Address']

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Type False

Export a ResultSet from asking a question as CSV with false for header_add_type

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
```

```
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_add_type"] = False

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Type True

Export a ResultSet from asking a question as CSV with true for header_add_type

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

```

```
# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_add_type"] = True

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
```

```
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Sensor False

Export a ResultSet from asking a question as CSV with false for header_add_sensor

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
```

```
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_add_sensor"] = False

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export into kwargs
kwargs['obj'] = response['question_results']

# export the object to a string
# (we could just as easily export to a file using export_to_report_file)
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
out = handler.export_obj(**kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
```

```
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the export_str returned from export_obj():"
print out
```

Export ResultSet CSV Sensor True

Export a ResultSet from asking a question as CSV with true for header_add_sensor

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1
```

```
43
44  # optional, use a debug format for the logging output (uses two lines per log entry)
45  handler_args['debugformat'] = False
46
47  # optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
48  # very useful for capturing the full exchange of XML requests and responses
49  handler_args['record_all_requests'] = True
50
51  # instantiate a handler using all of the arguments in the handler_args dictionary
52  print "...CALLING: pytan.handler() with args: {}".format(handler_args)
53  handler = pytan.Handler(**handler_args)
54
55  # print out the handler string
56  print "...OUTPUT: handler string: {}".format(handler)
57
58  # setup the arguments for the handler() class
59  kwargs = {}
60  kwargs["export_format"] = u'csv'
61  kwargs["header_add_sensor"] = True
62
63  # setup the arguments for handler.ask()
64  ask_kwargs = {
65      'qtype': 'manual',
66      'sensors': [
67          "Computer Name", "IP Route Details", "IP Address",
68          'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*Shared.*}',
69      ],
70  }
71
72  # ask the question that will provide the resultset that we want to use
73  print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
74  response = handler.ask(**ask_kwargs)
75
76  # store the resultset object as the obj we want to export into kwargs
77  kwargs['obj'] = response['question_results']
78
79  # export the object to a string
80  # (we could just as easily export to a file using export_to_report_file)
81  print "...CALLING: handler.export_obj() with args {}".format(kwargs)
82  out = handler.export_obj(**kwargs)
83
84  # trim the output if it is more than 15 lines long
85  if len(out.splitlines()) > 15:
86      out = out.splitlines()[0:15]
87      out.append('..trimmed for brevity..')
88      out = '\n'.join(out)
89
90  print "...OUTPUT: print the export_str returned from export_obj():"
91  print out
```

1.6.7 PyTan API Valid Get Object Examples

All of the PyTan API examples for Valid Get Object

Get Action By Id

Get an action by id

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'action'
kwargs["id"] = 1

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Question By Id

Get a question by id

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'question'
kwargs["id"] = 1

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response
```

```

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Saved Question By Names

Get two saved questions by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

```

```
# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'saved_question'
kwargs["name"] = [u'Installed Applications', u'Computer Name']

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Userrole By Id

Get a user role by id.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'userrole'
kwargs["id"] = 1

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Leader Clients

Get all clients that are Leader status

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'client'
kwargs["status"] = u'Leader'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response
```

```

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Setting By Name

Get a system setting by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

```

```
# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'setting'
kwargs["name"] = u'control_address'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get User By Name

Get a user by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'user'
kwargs["name"] = u'Administrator'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Sensor By Id

Get a sensor by id

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'sensor'
kwargs["id"] = 1

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response
```

```

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Sensor By Mixed

Get multiple sensors by id, name, and hash

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

```

```
# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'sensor'
kwargs["hash"] = [u'322086833']
kwargs["name"] = [u'Computer Name']
kwargs["id"] = [1, 2]

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Whitelisted Url By Id

Get a whitelisted url by id

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'whitelisted_url'
kwargs["id"] = 1

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Group By Name

Get a group by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'group'
kwargs["name"] = u'All Computers'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response
```

```

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Sensor By Hash

Get a sensor by hash

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

```

```
# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'sensor'
kwargs["hash"] = u'322086833'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Package By Name

Get a package by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'package'
kwargs["name"] = u'Distribute Tanium Standard Utilities'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Sensor By Names

Get multiple sensors by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'sensor'
kwargs["name"] = [u'Computer Name', u'Action Statuses']

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response
```

```

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Saved Question By Name

Get saved question by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

```

```
# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'saved_question'
kwargs["name"] = u'Installed Applications'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get User By Id

Get a user by id

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'user'
kwargs["id"] = 1

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Sensor By Name

Get a sensor by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'sensor'
kwargs["name"] = u'Computer Name'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response
```

```

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get Saved Action By Name

Get a saved action by name

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

```

```
# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'saved_action'
kwargs["name"] = u'Distribute Tanium Standard Utilities'

print "...CALLING: handler.get with args: {}".format(kwargs)
response = handler.get(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Users

Get all users

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'user'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Saved Actions

Get all saved actions

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
```

```
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'saved_action'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
```

```
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Settings

Get all system settings

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}
```

```
33
34  # establish our connection info for the Tanium Server
35  handler_args['username'] = "Administrator"
36  handler_args['password'] = "Tanium2015!"
37  handler_args['host'] = "10.0.1.240"
38  handler_args['port'] = "443"  # optional
39
40  # optional, level 0 is no output except warnings/errors
41  # level 1 through 12 are more and more verbose
42  handler_args['loglevel'] = 1
43
44  # optional, use a debug format for the logging output (uses two lines per log entry)
45  handler_args['debugformat'] = False
46
47  # optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
48  # very useful for capturing the full exchange of XML requests and responses
49  handler_args['record_all_requests'] = True
50
51  # instantiate a handler using all of the arguments in the handler_args dictionary
52  print "...CALLING: pytan.handler() with args: {}".format(handler_args)
53  handler = pytan.Handler(**handler_args)
54
55  # print out the handler string
56  print "...OUTPUT: handler string: {}".format(handler)
57
58  # setup the arguments for the handler() class
59  kwargs = {}
60  kwargs["objtype"] = u'setting'
61
62  print "...CALLING: handler.get_all with args: {}".format(kwargs)
63  response = handler.get_all(**kwargs)
64
65  print "...OUTPUT: Type of response: ", type(response)
66
67  print "...OUTPUT: print of response:"
68  print response
69
70  # call the export_obj() method to convert response to JSON and store it in out
71  export_kwargs = {}
72  export_kwargs['obj'] = response
73  export_kwargs['export_format'] = 'json'
74
75  print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
76  out = handler.export_obj(**export_kwargs)
77
78  # trim the output if it is more than 15 lines long
79  if len(out.splitlines()) > 15:
80      out = out.splitlines()[0:15]
81      out.append('..trimmed for brevity..')
82      out = '\n'.join(out)
83
84  print "...OUTPUT: print the objects returned in JSON format:"
85  print out
```

Get All Saved Questions

Get all saved questions

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```python
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

```

```
# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'saved_question'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Userroless

Get all user roles

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
```

```
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'userrole'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

```

```
print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Questions

Get all questions

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
```

```
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'question'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Groups

Get all groups

- STDOUT from Example Python Code
- STDERR from Example Python Code

- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

```

```
# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'group'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Sensors

Get all sensors

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
```

```
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'sensor'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

```

```
# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Whitelisted Urls

Get all whitelisted urls

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

```

```
# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'whitelisted_url'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Clients

Get all clients

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
```

```
kwargs = {}
kwargs["objtype"] = u'client'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Packages

Get all packages

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
```

```
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'package'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
```

```
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

Get All Actions

Get all actions

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
```

```
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'action'

print "...CALLING: handler.get_all with args: {}".format(kwargs)
response = handler.get_all(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: print of response:"
print response

# call the export_obj() method to convert response to JSON and store it in out
export_kwargs = {}
export_kwargs['obj'] = response
export_kwargs['export_format'] = 'json'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: print the objects returned in JSON format:"
print out
```

1.6.8 PyTan API Valid Questions Examples

All of the PyTan API examples for Valid Questions

Ask Manual Question Simple Multiple Sensors

Ask a manual question using human strings by referencing the name of multiple sensors in a list.

No sensor filters, sensor parameters, sensor filter options, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```python
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

```

```
# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = [u'Computer Name', u'Installed Applications']
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Simple Single Sensor

Ask a manual question using human strings by referencing the name of a single sensor in a string.

No sensor filters, sensor parameters, sensor filter options, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

```

```
# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Computer Name'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
```

```
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Multiple Sensors Identified By Name

Ask a manual question using human strings by referencing the name of multiple sensors and providing a selector that tells pytan explicitly that we are providing a name of a sensor.

No sensor filters, sensor parameters, sensor filter options, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

```

```
# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = [u'name:Computer Name', u'name:Installed Applications']
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

```

```
# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Sensor With Parameters And Some Supplied Parameters

Ask a manual question using human strings by referencing the name of a single sensor that takes parameters, but supplying only two of the four parameters that are used by the sensor (and letting pytan automatically determine the appropriate default value for those parameters which require a value and none was supplied).

No sensor filters, sensor parameters, sensor filter options, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
```

```
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Folder Name Search with RegEx Match{dirname=Program Files,regex=Microsoft.*}'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Multiple Sensors With Parameters And Some Supplied Parameters

Ask a manual question using human strings by referencing the name of multiple sensors, one that takes parameters, but supplying only two of the four parameters that are used by the sensor (and letting pytan automatically determine the appropriate default value for those parameters which require a value and none was supplied), and one that does not take parameters.

No sensor filters, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False
```

```
46
47  # optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
48  # very useful for capturing the full exchange of XML requests and responses
49  handler_args['record_all_requests'] = True
50
51  # instantiate a handler using all of the arguments in the handler_args dictionary
52  print "...CALLING: pytan.handler() with args: {}".format(handler_args)
53  handler = pytan.Handler(**handler_args)
54
55  # print out the handler string
56  print "...OUTPUT: handler string: {}".format(handler)
57
58  # setup the arguments for the handler() class
59  kwargs = {}
60  kwargs["sensors"] = [u'Folder Name Search with RegEx Match{dirname=Program Files,regex=Microsoft.*}',
61   u'Computer Name']
62  kwargs["qtype"] = u'manual'
63
64  print "...CALLING: handler.ask with args: {}".format(kwargs)
65  response = handler.ask(**kwargs)
66
67  print "...OUTPUT: Type of response: ", type(response)
68
69  print "...OUTPUT: Pretty print of response:"
70  print pprint.pformat(response)
71
72  print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
73  print response['question_object'].query_text
74
75  # call the export_obj() method to convert response to CSV and store it in out
76  export_kwargs = {}
77  export_kwargs['obj'] = response['question_results']
78  export_kwargs['export_format'] = 'csv'
79
80  print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
81  out = handler.export_obj(**export_kwargs)
82
83  # trim the output if it is more than 15 lines long
84  if len(out.splitlines()) > 15:
85      out = out.splitlines()[0:15]
86      out.append('..trimmed for brevity..')
87      out = '\n'.join(out)
88
89  print "...OUTPUT: CSV Results of response: "
90  print out
```

Ask Manual Question Sensor Without Parameters And Supplied Parameters

Ask a manual question using human strings by referencing the name of a single sensor that does NOT take parameters, but supplying parameters anyways (which will be ignored since the sensor does not take parameters).

No sensor filters, sensor filter options, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
```

```
kwargs = {}
kwargs["sensors"] = u'Computer Name{fake=Dweedle}'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Sensor With Parameters And No Supplied Parameters

Ask a manual question using human strings by referencing the name of a single sensor that takes parameters, but not supplying any parameters (and letting pytan automatically determine the appropriate default value for those parameters which require a value).

No sensor filters, sensor parameters, sensor filter options, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
```

```
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Folder Name Search with RegEx Match'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

```

```
print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Sensor With Parameters And Filter

Ask a manual question using human strings by referencing the name of a single sensor that takes parameters, but supplying only two of the four parameters that are used by the sensor.

Also supply a sensor filter that limits the column data that is shown to values that match the regex '.*Shared.*'.

No sensor filter options, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
```

```
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Folder Name Search with RegEx Match{dirname=Program Files,regex=Microsoft.*}, t
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
```

```
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Sensor With Filter And 2 Options

Ask a manual question using human strings by referencing the name of a single sensor.

Also supply a sensor filter that limits the column data that is shown to values that contain Windows (which is short hand for regex match against .*Windows.*).

Also supply filter options that re-fetches any cached data that is older than 3600 seconds and treats the values as type string.

No question filters or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

```

```
# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Operating System, that contains:Windows, opt:max_data_age:3600, opt:value_type
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Sensor With Filter

Ask a manual question using human strings by referencing the name of a single sensor.

Also supply a sensor filter that limits the column data that is shown to values that contain Windows (which is short hand for regex match against .*Windows.*).

No sensor parameters, sensor filter options, question filters or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False
```

```
46
47  # optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
48  # very useful for capturing the full exchange of XML requests and responses
49  handler_args['record_all_requests'] = True
50
51  # instantiate a handler using all of the arguments in the handler_args dictionary
52  print "...CALLING: pytan.handler() with args: {}".format(handler_args)
53  handler = pytan.Handler(**handler_args)
54
55  # print out the handler string
56  print "...OUTPUT: handler string: {}".format(handler)
57
58  # setup the arguments for the handler() class
59  kwargs = {}
60  kwargs["sensors"] = u'Operating System, that contains:Windows'
61  kwargs["qtype"] = u'manual'
62
63  print "...CALLING: handler.ask with args: {}".format(kwargs)
64  response = handler.ask(**kwargs)
65
66  print "...OUTPUT: Type of response: ", type(response)
67
68  print "...OUTPUT: Pretty print of response:"
69  print pprint.pformat(response)
70
71  print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
72  print response['question_object'].query_text
73
74  # call the export_obj() method to convert response to CSV and store it in out
75  export_kwargs = {}
76  export_kwargs['obj'] = response['question_results']
77  export_kwargs['export_format'] = 'csv'
78
79  print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
80  out = handler.export_obj(**export_kwargs)
81
82  # trim the output if it is more than 15 lines long
83  if len(out.splitlines()) > 15:
84      out = out.splitlines()[0:15]
85      out.append('..trimmed for brevity..')
86      out = '\n'.join(out)
87
88  print "...OUTPUT: CSV Results of response: "
89  print out
```

Ask Manual Question Sensor With Parameters And Filter And Options

Ask a manual question using human strings by referencing the name of a single sensor that takes parameters, but supplying only two of the four parameters that are used by the sensor.

Also supply a sensor filter that limits the column data that is shown to values that match the regex '.*Shared.*', and a sensor filter option that re-fetches any cached data that is older than 3600 seconds.

No question filters or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code

- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

```

```
# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Folder Name Search with RegEx Match{dirname=Program Files,regex=Microsoft.*}, t
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Sensor With Filter And 3 Options

Ask a manual question using human strings by referencing the name of a single sensor.

Also supply a sensor filter that limits the column data that is shown to values that contain Windows (which is short hand for regex match against .*Windows.*).

Also supply filter options that re-fetches any cached data that is older than 3600 seconds, matches all values supplied in the filter, and ignores case for any value match of the filter.

No sensor paramaters, question filters, or question options supplied.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
```

```
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Operating System, that contains:Windows, opt:match_all_values, opt:ignore_case,
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)
```

```

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Complex Query1

Ask a manual question using human strings by referencing the name of a two sensors sensor.

Supply 3 parameters for the second sensor, one of which is not a valid parameter (and will be ignored).

Supply one option to the second sensor.

Supply two question filters that limit the rows returned in the result to computers that match the sensor Operating System that contains Windows and does not contain Windows.

Supply two question options that 'or' the two question filters and ignore the case of any values while matching the question filters.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
```

```
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["question_filters"] = [u'Operating System, that contains:Windows',
 u'Operating System, that does not contain:Windows']
kwargs["sensors"] = [u'Computer Name',
 u'Folder Name Search with RegEx Match{dirname=Program Files,regex=Microsoft.*, invalidparam=test}, t
kwargs["question_options"] = [u'ignore_case', u'or']
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

```

```
print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Complex Query2

This is another complex query that gets the Computer Name and Last Logged in User and Installed Applications that contains Google Search or Google Chrome and limits the rows that are displayed to computers that contain the Installed Applications of Google Search or Google Chrome

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]
```

```

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["question_filters"] = [u'Installed Applications, that regex match:.*Google (Search|Chrome).*']
kwargs["sensors"] = [u'Computer Name',
 u'Last Logged In User',
 u'Installed Applications, that regex match:.*Google (Search|Chrome).*']
kwargs["question_options"] = [u'ignore_case', u'or']
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)
```

```

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Manual Question Sensor Complex

This provides an example for asking a manual question without using human strings.

It uses the Computer Name and Folder Name Search with RegEx Match sensors.

The second sensor has a single parameter, dirname, with a value of 'Program Files'.

The second sensor also has 3 sensor filter options that set the max data age to 3600 seconds, does NOT ignore case, and treats all values as string.

There is also a question filter supplied that limits the rows that are displayed to computers that match an Operating System that contains Windows, and has 3 question filter options supplied that set the max data age to 3600 seconds, does NOT ignore case, and uses 'and' to join all question filters.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
```

```
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["question_filter_defs"] = [{u'filter': {u'not_flag': 0,
              u'operator': u'RegexMatch',
              u'value': u'.*Windows.*'},
  u'name': u'Operating System'}]
kwargs["sensor_defs"] = [u'Computer Name',
 {u'filter': {u'not_flag': 0,
              u'operator': u'RegexMatch',
              u'value': u'.*Shared.*'},
  u'name': u'Folder Name Search with RegEx Match',
  u'options': {u'ignore_case_flag': 0,
               u'max_age_seconds': 3600,
               u'value_type': u'string'},
  u'params': {u'dirname': u'Program Files'}}]
kwargs["question_option_defs"] = {u'and_flag': 0, u'ignore_case_flag': 0, u'max_age_seconds': 3600}
kwargs["qtype"] = u'_manual'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

```

```
# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

1.6.9 PyTan API Valid Saved Questions Examples

All of the PyTan API examples for Valid Saved Questions

Ask Saved Question Refresh Data

Ask a saved question and refresh the data for the saved question (asks a new question)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]
```

```
27
28  # import pytan
29  import pytan
30
31  # create a dictionary of arguments for the pytan handler
32  handler_args = {}
33
34  # establish our connection info for the Tanium Server
35  handler_args['username'] = "Administrator"
36  handler_args['password'] = "Tanium2015!"
37  handler_args['host'] = "10.0.1.240"
38  handler_args['port'] = "443"  # optional
39
40  # optional, level 0 is no output except warnings/errors
41  # level 1 through 12 are more and more verbose
42  handler_args['loglevel'] = 1
43
44  # optional, use a debug format for the logging output (uses two lines per log entry)
45  handler_args['debugformat'] = False
46
47  # optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
48  # very useful for capturing the full exchange of XML requests and responses
49  handler_args['record_all_requests'] = True
50
51  # instantiate a handler using all of the arguments in the handler_args dictionary
52  print "...CALLING: pytan.handler() with args: {}".format(handler_args)
53  handler = pytan.Handler(**handler_args)
54
55  # print out the handler string
56  print "...OUTPUT: handler string: {}".format(handler)
57
58  # setup the arguments for the handler() class
59  kwargs = {}
60  kwargs["refresh_data"] = True
61  kwargs["qtype"] = u'saved'
62  kwargs["name"] = u'Installed Applications'
63
64  print "...CALLING: handler.ask with args: {}".format(kwargs)
65  response = handler.ask(**kwargs)
66
67  print "...OUTPUT: Type of response: ", type(response)
68
69  print "...OUTPUT: Pretty print of response:"
70  print pprint.pformat(response)
71
72  print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
73  print response['question_object'].query_text
74
75  # call the export_obj() method to convert response to CSV and store it in out
76  export_kwargs = {}
77  export_kwargs['obj'] = response['question_results']
78  export_kwargs['export_format'] = 'csv'
79
80  print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
81  out = handler.export_obj(**export_kwargs)
82
83  # trim the output if it is more than 15 lines long
84  if len(out.splitlines()) > 15:
```

```
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Saved Question By Name

Ask a saved question by referencing the name of a saved question in a string.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
```

```
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["qtype"] = u'saved'
kwargs["name"] = u'Installed Applications'

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

Ask Saved Question By Name In List

Ask a saved question by referencing the name of a saved question in a list of strings.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
```

```
kwargs = {}
kwargs["qtype"] = u'saved'
kwargs["name"] = [u'Installed Applications']

print "...CALLING: handler.ask with args: {}".format(kwargs)
response = handler.ask(**kwargs)

print "...OUTPUT: Type of response: ", type(response)

print "...OUTPUT: Pretty print of response:"
print pprint.pformat(response)

print "...OUTPUT: Equivalent Question if it were to be asked in the Tanium Console: "
print response['question_object'].query_text

# call the export_obj() method to convert response to CSV and store it in out
export_kwargs = {}
export_kwargs['obj'] = response['question_results']
export_kwargs['export_format'] = 'csv'

print "...CALLING: handler.export_obj() with args {}".format(export_kwargs)
out = handler.export_obj(**export_kwargs)

# trim the output if it is more than 15 lines long
if len(out.splitlines()) > 15:
    out = out.splitlines()[0:15]
    out.append('..trimmed for brevity..')
    out = '\n'.join(out)

print "...OUTPUT: CSV Results of response: "
print out
```

1.6.10 PyTan API Invalid Create Object Examples

All of the PyTan API examples for Invalid Create Object

Invalid Create Sensor

Create a sensor (Unsupported!)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
```

```
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["unsupported"] = True

print "...CALLING: handler.create_sensor() with args: {}".format(kwargs)
try:
    handler.create_sensor(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

1.6.11 PyTan API Invalid Create Object From JSON Examples

All of the PyTan API examples for Invalid Create Object From JSON

Invalid Create Saved Action From JSON

Create a saved action from json (not supported!)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False
```

```

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'saved_action'
get_kwargs["name"] = u'Distribute Tanium Standard Utilities'

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'saved_action'
create_kwargs['json_file'] = json_file

# call the handler with the create_from_json method, passing in kwargs for arguments
print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
try:
    response = handler.create_from_json(**create_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Create Client From JSON

Create a client from json (not supported!)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
```

```
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'client'
get_kwargs["status"] = u'Leader'
```

```

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'client'
create_kwargs['json_file'] = json_file

# call the handler with the create_from_json method, passing in kwargs for arguments
print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
try:
    response = handler.create_from_json(**create_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Create Userrole From JSON

Create a user role from json (not supported!)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
```

```
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'userrole'
get_kwargs["name"] = u'Administrator'

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
```

```
create_kwargs['objtype'] = u'userrole'
create_kwargs['json_file'] = json_file

# call the handler with the create_from_json method, passing in kwargs for arguments
print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
try:
    response = handler.create_from_json(**create_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Create Setting From JSON

Create a setting from json (not supported!)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
```

```
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler.get() method
get_kwargs = {}
get_kwargs["objtype"] = u'setting'
get_kwargs["id"] = 1

# get objects to use as an export to JSON file
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
orig_objs = handler.get(**get_kwargs)

# export orig_objs to a json file
export_kwargs = {}
export_kwargs['obj'] = orig_objs
export_kwargs['export_format'] = 'json'
export_kwargs['report_dir'] = tempfile.gettempdir()

print "...CALLING: handler.export_to_report_file() with args: {}".format(export_kwargs)
json_file, results = handler.export_to_report_file(**export_kwargs)

# create the object from the exported JSON file
create_kwargs = {}
create_kwargs['objtype'] = u'setting'
create_kwargs['json_file'] = json_file

# call the handler with the create_from_json method, passing in kwargs for arguments
print "...CALLING: handler.create_from_json() with args {}".format(create_kwargs)
try:
    response = handler.create_from_json(**create_kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

1.6.12 PyTan API Invalid Deploy Action Examples

All of the PyTan API examples for Invalid Deploy Action

Invalid Deploy Action Run False

Deploy an action without run=True, which will only run the pre-deploy action question that matches action_filters, export the results to a file, and raise a RunFalse exception

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

```

```
# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["report_dir"] = u'/tmp'
kwargs["package"] = u'Distribute Tanium Standard Utilities'

print "...CALLING: handler.deploy_action() with args: {}".format(kwargs)
try:
    handler.deploy_action(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.RunFalse
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Deploy Action Package Help

Have deploy_action() return the help for package

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
```

```
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["package_help"] = True

print "...CALLING: handler.deploy_action() with args: {}".format(kwargs)
try:
    handler.deploy_action(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.PytanHelp
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Deploy Action Package

Deploy an action using a non-existing package.

- STDOUT from Example Python Code
- STDERR from Example Python Code

- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

```

```
# setup the arguments for the handler() class
kwargs = {}
kwargs["run"] = True
kwargs["package"] = u'Invalid Package'

print "...CALLING: handler.deploy_action() with args: {}".format(kwargs)
try:
    handler.deploy_action(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Deploy Action Options Help

Have deploy_action() return the help for options

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
```

```
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["options_help"] = True

print "...CALLING: handler.deploy_action() with args: {}".format(kwargs)
try:
    handler.deploy_action(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.PytanHelp
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Deploy Action Empty Package

Deploy an action using an empty package string.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
```

```
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["run"] = True
kwargs["package"] = u''

print "...CALLING: handler.deploy_action() with args: {}".format(kwargs)
try:
    handler.deploy_action(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HumanParserError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Deploy Action Filters Help

Have deploy_action() return the help for filters

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["filters_help"] = True

print "...CALLING: handler.deploy_action() with args: {}".format(kwargs)
try:
    handler.deploy_action(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.PytanHelp
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Deploy Action Missing Parameters

Deploy an action using a package that requires parameters but do not supply any parameters.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
```

```
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["run"] = True
kwargs["package"] = u'Custom Tagging - Add Tags'

print "...CALLING: handler.deploy_action() with args: {}".format(kwargs)
try:
    handler.deploy_action(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

1.6.13 PyTan API Invalid Export Basetype Examples

All of the PyTan API examples for Invalid Export Basetype

Invalid Export Basetype CSV Bad Explode Type

Export a BaseType from getting objects using a bad explode_json_string_values

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
```

```
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["explode_json_string_values"] = u'bad'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to use
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export Basetype CSV Bad Sort Sub Type

Export a BaseType from getting objects using a bad header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

```

```
# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = [[]]

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to use
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
```

```
kwargs['obj'] = response

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export Basetype CSV Bad Sort Type

Export a BaseType from getting objects using a bad header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
```

```
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = u'bad'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to use
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export Basetype XML Bad Minimal Type

Export a BaseType from getting objects using a bad minimal

- STDOUT from Example Python Code

- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
```

```
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'xml'
kwargs["minimal"] = u'bad'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to use
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export Basetype JSON Bad Include Type

Export a BaseType from getting objects using a bad include_type

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
```

```
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'
kwargs["include_type"] = u'bad'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to use
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
```

```
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export Basetype JSON Bad Explode Type

Export a BaseType from getting objects using a bad explode_json_string_values

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

```

```
# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'json'
kwargs["explode_json_string_values"] = u'bad'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to use
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export Basetype Bad Format

Export a BaseType from getting objects using a bad export_format

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'bad'

# setup the arguments for handler.get()
get_kwargs = {
    'name': [
        "Computer Name", "IP Route Details", "IP Address",
        'Folder Name Search with RegEx Match',
    ],
    'objtype': 'sensor',
}

# get the objects that will provide the basetype that we want to use
print "...CALLING: handler.get() with args: {}".format(get_kwargs)
response = handler.get(**get_kwargs)

# store the basetype object as the obj we want to export
kwargs['obj'] = response

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

1.6.14 PyTan API Invalid Export ResultSet Examples

All of the PyTan API examples for Invalid Export ResultSet

Invalid Export ResultSet CSV Bad Sort Sub Type

Export a ResultSet from asking a question using a bad header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback
```

```
7
8  # disable python from generating a .pyc file
9  sys.dont_write_bytecode = True
10
11 # change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
12 pytan_loc = "~/gh/pytan"
13 pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')
14
15 # Determine our script name, script dir
16 my_file = os.path.abspath(sys.argv[0])
17 my_dir = os.path.dirname(my_file)
18
19 # try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
20 parent_dir = os.path.dirname(my_dir)
21 pytan_root_dir = os.path.dirname(parent_dir)
22 lib_dir = os.path.join(pytan_root_dir, 'lib')
23
24 # add pytan_loc and lib_dir to the PYTHONPATH variable
25 path_adds = [lib_dir, pytan_static_path]
26 [sys.path.append(aa) for aa in path_adds if aa not in sys.path]
27
28 # import pytan
29 import pytan
30
31 # create a dictionary of arguments for the pytan handler
32 handler_args = {}
33
34 # establish our connection info for the Tanium Server
35 handler_args['username'] = "Administrator"
36 handler_args['password'] = "Tanium2015!"
37 handler_args['host'] = "10.0.1.240"
38 handler_args['port'] = "443"  # optional
39
40 # optional, level 0 is no output except warnings/errors
41 # level 1 through 12 are more and more verbose
42 handler_args['loglevel'] = 1
43
44 # optional, use a debug format for the logging output (uses two lines per log entry)
45 handler_args['debugformat'] = False
46
47 # optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
48 # very useful for capturing the full exchange of XML requests and responses
49 handler_args['record_all_requests'] = True
50
51 # instantiate a handler using all of the arguments in the handler_args dictionary
52 print "...CALLING: pytan.handler() with args: {}".format(handler_args)
53 handler = pytan.Handler(**handler_args)
54
55 # print out the handler string
56 print "...OUTPUT: handler string: {}".format(handler)
57
58 # setup the arguments for the handler() class
59 kwargs = {}
60 kwargs["export_format"] = u'csv'
61 kwargs["header_sort"] = [[]]
62
63 # setup the arguments for handler.ask()
64 ask_kwargs = {
```

```
    'qtype': 'manual',
    'sensors': [
        "Computer Name"
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export
kwargs['obj'] = response['question_results']

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export ResultSet CSV Bad Sort Type

Export a ResultSet from asking a question using a bad header_sort

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
```

```
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["header_sort"] = u'bad'

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name"
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export
kwargs['obj'] = response['question_results']

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
```

```
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export ResultSet CSV Bad Expand Type

Export a ResultSet from asking a question using a bad expand_grouped_columns

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
```

```
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["expand_grouped_columns"] = u'bad'

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name"
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export
kwargs['obj'] = response['question_results']

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export ResultSet CSV Bad Sensors Sub Type

Export a ResultSet from asking a question using a bad sensors

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
```

```
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'csv'
kwargs["sensors"] = [[]]
kwargs["header_add_sensor"] = True

```

```
# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name"
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export
kwargs['obj'] = response['question_results']

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Export ResultSet Bad Format

Export a ResultSet from asking a question using a bad export_format

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

```

```
# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["export_format"] = u'bad'

# setup the arguments for handler.ask()
ask_kwargs = {
    'qtype': 'manual',
    'sensors': [
        "Computer Name"
    ],
}

# ask the question that will provide the resultset that we want to use
print "...CALLING: handler.ask() with args {}".format(ask_kwargs)
response = handler.ask(**ask_kwargs)

# store the resultset object as the obj we want to export
kwargs['obj'] = response['question_results']

# export the object to a string
print "...CALLING: handler.export_obj() with args {}".format(kwargs)
try:
    handler.export_obj(**kwargs)
except Exception as e:
```

```
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

1.6.15 PyTan API Invalid Get Object Examples

All of the PyTan API examples for Invalid Get Object

Invalid Get Action Single By Name

Get an action by name (name is not a supported selector for action)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional
```

```

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'action'
kwargs["name"] = u'Distribute Tanium Standard Utilities'

print "...CALLING: handler.get() with args: {}".format(kwargs)
try:
    handler.get(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Get Question By Name

Get a question by name (name is not a supported selector for question)

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
```

```
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["objtype"] = u'question'
kwargs["name"] = u'dweedle'

print "...CALLING: handler.get() with args: {}".format(kwargs)
try:
    handler.get(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

1.6.16 PyTan API Invalid Questions Examples

All of the PyTan API examples for Invalid Questions

Invalid Ask Manual Question Sensor Help

Have ask_manual() return the help for sensors

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False
```

```

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["qtype"] = u'manual'
kwargs["sensors_help"] = True

print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.PytanHelp
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Ask Manual Question Filter Help

Have ask_manual() return the help for filters

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')
```

```

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["filters_help"] = True
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.PytanHelp
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Ask Manual Question Option Help

Have ask_manual() return the help for options

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
```

```
kwargs = {}
kwargs["options_help"] = True
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.PytanHelp
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Ask Manual Question Bad Filter

Ask a question using an invalid filter.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
```

```
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Computer name, that does not meet:little'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HumanParserError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Ask Manual Question Bad Sensorname

Ask a question using a sensor that does not exist

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
```

```
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Dweedle Dee and Dum'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HandlerError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Ask Manual Question Too Many Parameter Blocks

Ask a question that supplies too many parameter blocks ({}).

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

```

```
# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Folder Name Search with RegEx Match{dirname=Program Files,regex=.*}{}'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HumanParserError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Ask Manual Question Bad Option

Ask a question using an invalid option.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

```

```
# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Operating system, opt:bad'
kwargs["qtype"] = u'manual'

print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HumanParserError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

Invalid Ask Manual Question Missing Parameter Split

Ask a question with parameters that are missing a splitter (=) to designate the key from value.

- STDOUT from Example Python Code
- STDERR from Example Python Code
- Example Python Code

```
# import the basic python packages we need
import os
import sys
import tempfile
```

```
import pprint
import traceback

# disable python from generating a .pyc file
sys.dont_write_bytecode = True

# change me to the path of pytan if this script is not running from EXAMPLES/PYTAN_API
pytan_loc = "~/gh/pytan"
pytan_static_path = os.path.join(os.path.expanduser(pytan_loc), 'lib')

# Determine our script name, script dir
my_file = os.path.abspath(sys.argv[0])
my_dir = os.path.dirname(my_file)

# try to automatically determine the pytan lib directory by assuming it is in '../../lib/'
parent_dir = os.path.dirname(my_dir)
pytan_root_dir = os.path.dirname(parent_dir)
lib_dir = os.path.join(pytan_root_dir, 'lib')

# add pytan_loc and lib_dir to the PYTHONPATH variable
path_adds = [lib_dir, pytan_static_path]
[sys.path.append(aa) for aa in path_adds if aa not in sys.path]

# import pytan
import pytan

# create a dictionary of arguments for the pytan handler
handler_args = {}

# establish our connection info for the Tanium Server
handler_args['username'] = "Administrator"
handler_args['password'] = "Tanium2015!"
handler_args['host'] = "10.0.1.240"
handler_args['port'] = "443"  # optional

# optional, level 0 is no output except warnings/errors
# level 1 through 12 are more and more verbose
handler_args['loglevel'] = 1

# optional, use a debug format for the logging output (uses two lines per log entry)
handler_args['debugformat'] = False

# optional, this saves all response objects to handler.session.ALL_REQUESTS_RESPONSES
# very useful for capturing the full exchange of XML requests and responses
handler_args['record_all_requests'] = True

# instantiate a handler using all of the arguments in the handler_args dictionary
print "...CALLING: pytan.handler() with args: {}".format(handler_args)
handler = pytan.Handler(**handler_args)

# print out the handler string
print "...OUTPUT: handler string: {}".format(handler)

# setup the arguments for the handler() class
kwargs = {}
kwargs["sensors"] = u'Computer Name{Dweedle}'
kwargs["qtype"] = u'manual'

```

```
print "...CALLING: handler.ask() with args: {}".format(kwargs)
try:
    handler.ask(**kwargs)
except Exception as e:
    print "...EXCEPTION: {}".format(e)
    # this should throw an exception of type: pytan.exceptions.HumanParserError
    # uncomment to see full exception
    # traceback.print_exc(file=sys.stdout)
```

1.7 SOAP API Examples

1.7.1 SOAP API Examples for Platform Version 6.5.314.4301

Each of these sections contains examples that show the HTTP request and response for each step in a given workflow.

Basic API Authentication

This is an example for how to authenticate against the SOAP API

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005930
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json

- HTTP Method: GET
- Elapsed Time: 0:00:00.013502
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6911-da6e5b707595c7b5e42d4738029d5c97256bb813fc843855cb9c675c54dacb06c8153557fb60ea1
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Create User

Create a user called API Test User

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005858
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007489
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6912-68d598b5538e59251cb4a35c2cb69fe3b5967d0713ca5031b766cabd6fd04d6126a9200d6843581
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002030
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6912-68d598b5538e59251cb4a35c2cb69fe3b5967d0713ca5031b766cabd6fd04d6126a9200d6843581
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001208
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6912-68d598b5538e59251cb4a35c2cb69fe3b5967d0713ca5031b766cabd6fd04d6126a9200d6843581
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue an AddObject to add a User object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.006900
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1792",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6912-68d598b5538e59251cb4a35c2cb69fe3b5967d0713ca5031b766cabd6fd04d6126a9200d6843581
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001727
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2897",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6912-68d598b5538e59251cb4a35c2cb69fe3b5967d0713ca5031b766cabd6fd04d6126a9200d6843581
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002022
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6912-68d598b5538e59251cb4a35c2cb69fe3b5967d0713ca5031b766cabd6fd04d6126a9200d6843581
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005434
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2846",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6912-68d598b5538e59251cb4a35c2cb69fe3b5967d0713ca5031b766cabd6fd04d6126a9200d6843581
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Package

Create a package called package49

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET

- Elapsed Time: 0:00:00.006884
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013476
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d9
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST

- Elapsed Time: 0:00:00.002518
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "510",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "667",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 4 - Issue a GetObject to get the full object of a sensor for inclusion in a question or action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001815
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "563",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 5 - Issue an AddObject to add a Group object for this package

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004569
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "647",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "762",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 6 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.014005
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "487",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
```

```
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue an AddObject to add a Group object for this package

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.007014
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "5193",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001833
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "5909",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001489
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "510",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013096
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "5858",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6913-78f5cc47430b95414a40a8125bc9afeaa40099d283b2a16b9d506393d17bde8bba85219dfd1e37d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Group

Create a group called All Windows Computers API Test

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006101
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.015088

- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6914-d31dcf0ab7d38ea2bd8256b9ae9e7d6bf5348b17cce76f3a15e01cf9f00047e409bc9697a8e93d02
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002629
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "534",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6914-d31dcf0ab7d38ea2bd8256b9ae9e7d6bf5348b17cce76f3a15e01cf9f00047e409bc9697a8e93d02
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "665",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 4 - Issue a GetObject to get the full object of specified sensors for inclusion in a group

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST

- Elapsed Time: 0:00:00.002210
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6914-d31dcf0ab7d38ea2bd8256b9ae9e7d6bf5348b17cce76f3a15e01cf9f00047e409bc9697a8e93d02
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue an AddObject to add a Group object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.007362
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "692",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6914-d31dcf0ab7d38ea2bd8256b9ae9e7d6bf5348b17cce76f3a15e01cf9f00047e409bc9697a8e93d02
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "762",
  "content-type": "text/xml;charset=UTF-8"
}
```

Step 6 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.011061
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "487",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6914-d31dcf0ab7d38ea2bd8256b9ae9e7d6bf5348b17cce76f3a15e01cf9f00047e409bc9697a8e93d02
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001725
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "534",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6914-d31dcf0ab7d38ea2bd8256b9ae9e7d6bf5348b17cce76f3a15e01cf9f00047e409bc9697a8e93d02
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002416
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1128",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6914-d31dcf0ab7d38ea2bd8256b9ae9e7d6bf5348b17cce76f3a15e01cf9f00047e409bc9697a8e93d02
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Whitelisted Url

Create a whitelisted url

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005672
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.012832
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6915-c6b9d055b32fff084601e262b1959939f54f4ee588a0003b5f75fc2a3e1fc64e2c66262f938b5a3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86179",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.277899
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "480",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6915-c6b9d055b32fff084601e262b1959939f54f4ee588a0003b5f75fc2a3e1fc64e2c66262f938b5a3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a WhitelistedURL object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.019582
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "698",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6915-c6b9d055b32fff084601e262b1959939f54f4ee588a0003b5f75fc2a3e1fc64e2c66262f938b5a3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "1020",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002186

- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "738",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6915-c6b9d055b32fff084601e262b1959939f54f4ee588a0003b5f75fc2a3e1fc64e2c66262f938b5a3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "991",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 6 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.298260
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "480",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6915-c6b9d055b32fff084601e262b1959939f54f4ee588a0003b5f75fc2a3e1fc64e2c66262f938b5a3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.004764
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "687",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6915-c6b9d055b32fff084601e262b1959939f54f4ee588a0003b5f75fc2a3e1fc64e2c66262f938b5a3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Package From JSON

Get a package object, add ' API TEST' to the name of the package object, delete any pre-existing package with the new name, then create a new package object with the new name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008272
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014468
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6916-f129723d3fb3582780ccc4638951752cca124ed745d06b4b45e6e4a925fa6be83376f5ba18a4097c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86179",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002893
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "499",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6916-f129723d3fb3582780ccc4638951752cca124ed745d06b4b45e6e4a925fa6be83376f5ba18a4097c
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001583
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "534",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6916-f129723d3fb3582780ccc4638951752cca124ed745d06b4b45e6e4a925fa6be83376f5ba18a4097c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005890
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "2407",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6916-f129723d3fb3582780ccc4638951752cca124ed745d06b4b45e6e4a925fa6be83376f5ba18a4097c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.008155
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2446",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6916-f129723d3fb3582780ccc4638951752cca124ed745d06b4b45e6e4a925fa6be83376f5ba18a4097c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001707
- Step 7 Request Body
- Step 7 Response Body

- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2262",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6916-f129723d3fb3582780ccc4638951752cca124ed745d06b4b45e6e4a925fa6be83376f5ba18a4097c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create User From JSON

Get a user object, add ‘ API TEST’ to the name of the user object, delete any pre-existing user with the new name, then create a new user object with the new name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005651
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013454
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6917-1e673660e061bc1418b184e82563f9713bd4f6e9fc9b9f1572dded518ff76b5c8ef44da01230c83
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86179",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001642
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "482",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6917-1e673660e061bc1418b184e82563f9713bd4f6e9fc9b9f1572dded518ff76b5c8ef44da01230c83
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Step 4 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001725
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6917-1e673660e061bc1418b184e82563f9713bd4f6e9fc9b9f1572dded518ff76b5c8ef44da01230c83
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 5 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003683
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2729",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6917-1e673660e061bc1418b184e82563f9713bd4f6e9fc9b9f1572dded518ff76b5c8ef44da01230c83
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004871
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2768",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6917-1e673660e061bc1418b184e82563f9713bd4f6e9fc9b9f1572dded518ff76b5c8ef44da01230c83
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002068
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "2780",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6917-1e673660e061bc1418b184e82563f9713bd4f6e9fc9b9f1572dded518ff76b5c8ef44da01230c83
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Saved Question From JSON

Get a saved question object, add ‘ API TEST’ to the name of the saved question object, delete any pre-existing saved question with the new name, then create a new saved question object with the new name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005834
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET

- Elapsed Time: 0:00:00.013184
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6918-c608bf977916033bd04dc9bf439ab3a54667c8810d210d950ca4fb630e8f009d9a843c7699f17c3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86179",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.011342
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "502",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6918-c608bf977916033bd04dc9bf439ab3a54667c8810d210d950ca4fb630e8f009d9a843c7699f17c3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.004730
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "543",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6918-c608bf977916033bd04dc9bf439ab3a54667c8810d210d950ca4fb630e8f009d9a843c7699f17c3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003435
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "11006",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6918-c608bf977916033bd04dc9bf439ab3a54667c8810d210d950ca4fb630e8f009d9a843c7699f17c3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 6 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005398
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "11047",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6918-c608bf977916033bd04dc9bf439ab3a54667c8810d210d950ca4fb630e8f009d9a843c7699f17c3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "831",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 7 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.007227
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "555",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6918-c608bf977916033bd04dc9bf439ab3a54667c8810d210d950ca4fb630e8f009d9a843c7699f17c3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
```

```
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Action From JSON

Get an action object, then create a new object from that (aka re-deploy an action)

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005778
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014138
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6919-207824e5cc87ff1c9c110ee9265a25a61182b7d49b2cabe1874429f40b39acb6836a7359b3ed2ac
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86181",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002205
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "486",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6919-207824e5cc87ff1c9c110ee9265a25a61182b7d49b2cabe1874429f40b39acb6836a7359b3ed2ac
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004685
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1357",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6919-207824e5cc87ff1c9c110ee9265a25a61182b7d49b2cabe1874429f40b39acb6836a7359b3ed2ac
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002555
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1368",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6919-207824e5cc87ff1c9c110ee9265a25a61182b7d49b2cabe1874429f40b39acb6836a7359b3ed2ac
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Sensor From JSON

Get a sensor object, add ‘ API TEST’ to the name of the sensor object, delete any pre-existing sensor with the new name, then create a new sensor object with the new name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006732
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007148
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6920-fc22898975af19a91cf1560c9c9cfea6669820e7ed35e7a1672ed77d3adb12add41e4acb3ba2883
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```

Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001364
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "507",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6920-fc22898975af19a91cf1560c9c9cfea6669820e7ed35e7a1672ed77d3adb12add41e4acb3ba2883
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001953
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "523",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6920-fc22898975af19a91cf1560c9c9cfea6669820e7ed35e7a1672ed77d3adb12add41e4acb3ba2883
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.007358
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1961",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6920-fc22898975af19a91cf1560c9c9cfea6669820e7ed35e7a1672ed77d3adb12add41e4acb3ba2883
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005388
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1977",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6920-fc22898975af19a91cf1560c9c9cfea6669820e7ed35e7a1672ed77d3adb12add41e4acb3ba2883
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "763",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 7 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.025365
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "488",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6920-fc22898975af19a91cf1560c9c9cfea6669820e7ed35e7a1672ed77d3adb12add41e4acb3ba2883
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Question From JSON

Get a question object, then create a new object from that (aka re-ask a question)

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005651

- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013641
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6921-b6479c197f6840955f0c6a1228688180964bb93eec39fcb7e7689fb0cfac468d4c943eebfd07277
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002235

- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "490",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6921-b6479c197f6840955f0c6a1228688180964bb93eec39fcb7e7689fb0cfac468d4c943eebfd07277
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005462
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2146",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6921-b6479c197f6840955f0c6a1228688180964bb93eec39fcb7e7689fb0cfac468d4c943eebfd07277
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.014726
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6921-b6479c197f6840955f0c6a1228688180964bb93eec39fcb7e7689fb0cfac468d4c943eebfd07277
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Create Whitelisted Url From JSON

Get a whitelisted url object, add ' API TEST' to the url_regex of the whitelisted url object, delete any pre-existing whitelisted url with the new url_regex, then create a new whitelisted url object with the new url_regex

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007380
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006730
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6922-c6924768f0749f065a818926b06de2a4da74c8f3d64ad752386557a9d4f379fb3163056e554fd945
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.287918
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "480",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6922-c6924768f0749f065a818926b06de2a4da74c8f3d64ad752386557a9d4f379fb3163056e554fd945
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.275546
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "480",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6922-c6924768f0749f065a818926b06de2a4da74c8f3d64ad752386557a9d4f379fb3163056e554fd945
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.009114
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "538",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6922-c6924768f0749f065a818926b06de2a4da74c8f3d64ad752386557a9d4f379fb3163056e554fd945
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "957",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 6 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.022226
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "575",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6922-c6924768f0749f065a818926b06de2a4da74c8f3d64ad752386557a9d4f379fb3163056e554fd945
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "866",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 7 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001966
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "589",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6922-c6924768f0749f065a818926b06de2a4da74c8f3d64ad752386557a9d4f379fb3163056e554fd945
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "837",
  "content-type": "text/xml;charset=UTF-8"
}
```


Create Group From JSON

Get a group object, add ' API TEST' to the name of the group object, delete any pre-existing group with the new name, then create a new group object with the new name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008859
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json

- HTTP Method: GET
- Elapsed Time: 0:00:00.014108
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6923-1b5896b8960f90f883e294e107cd909333673f42fa0e820a1e58cd246a1e64db99db50b01c7aa28a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003172
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "517",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6923-1b5896b8960f90f883e294e107cd909333673f42fa0e820a1e58cd246a1e64db99db50b01c7aa28a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "941",
  "content-type": "text/xml;charset=UTF-8"
}
```

Step 4 - Issue a GetObject to find the object to be deleted

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001771
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6923-1b5896b8960f90f883e294e107cd909333673f42fa0e820a1e58cd246a1e64db99db50b01c7aa28a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "952",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a DeleteObject to delete an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002377
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "620",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6923-1b5896b8960f90f883e294e107cd909333673f42fa0e820a1e58cd246a1e64db99db50b01c7aa28a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "950",
```

```
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 6 - Issue an AddObject to add an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003491
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "658",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6923-1b5896b8960f90f883e294e107cd909333673f42fa0e820a1e58cd246a1e64db99db50b01c7aa28a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "762",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 7 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.014759
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "487",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6923-1b5896b8960f90f883e294e107cd909333673f42fa0e820a1e58cd246a1e64db99db50b01c7aa28a
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "934",
  "content-type": "text/xml;charset=UTF-8"
}
```


Deploy Action Simple

Deploy an action using the package 'Distribute Tanium Standard Utilities' to all computers, wait for result data to be complete, and then get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005787
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007137
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86181",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a package for inclusion in an action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003097
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "581",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a list of SavedActions (6.5 logic)

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004732
- Step 4 Request Body

- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1493",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003382
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1523",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetObject to get the last action created for a SavedAction

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.002423
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "557",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject to get the package for an Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001937
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "600",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 8 - Issue a GetResultInfo on an Action to have the Server create a question that tracks the results for a Deployed Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003328
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetObject on the package for an action to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001953
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "625",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - ID 522: Issuing an AddObject of a Question object with no Selects and the same Group used by the Action object. The number of systems that should successfully run the Action will be taken from result_info.passed_count for the Question asked when all answers for the question have reported in.

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002457
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "525",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647aff
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 11 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002371
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001251
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 13 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002060
- Step 13 Request Body
- Step 13 Response Body

- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002729
- Step 14 Request Body
- Step 14 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1460",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 15 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.002933
- Step 15 Request Body
- Step 15 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 16 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002875
- Step 16 Request Body
- Step 16 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 17 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003173
- Step 17 Request Body
- Step 17 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1460",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 18 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003020
- Step 18 Request Body
- Step 18 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 19 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002767
- Step 19 Request Body
- Step 19 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647aff
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 20 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003202
- Step 20 Request Body
- Step 20 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
```

```
  "Connection": "keep-alive",
  "Content-Length": "1460",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 21 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002996
- Step 21 Request Body
- Step 21 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 22 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002978
- Step 22 Request Body

- Step 22 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 23 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003231
- Step 23 Request Body
- Step 23 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1460",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 24 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003172
- Step 24 Request Body
- Step 24 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 25 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002963
- Step 25 Request Body
- Step 25 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 26 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003103
- Step 26 Request Body
- Step 26 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1460",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647aff
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 27 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003306
- Step 27 Request Body
- Step 27 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 28 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003108
- Step 28 Request Body
- Step 28 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 29 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002511
- Step 29 Request Body

- Step 29 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1460",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 30 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002556
- Step 30 Request Body
- Step 30 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647affi
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 31 - Issue a GetResultData for an Action with the aggregate option set to False. This will return all of the Action Statuses for each computer that have run this Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002572
- Step 31 Request Body
- Step 31 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "580",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6924-eb9336c22ad9373c3520002c9e27872eb9f25aa1b7b74bb32d1005293ff1127f6ca5b39c0647aff
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Deploy Action Simple Without Results

Deploy an action using the package 'Distribute Tanium Standard Utilities' to all computers and do not wait for result data to be complete and do not get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008210
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
```

```
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014382
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86180",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a package for inclusion in an action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003270
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "581",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a list of SavedActions (6.5 logic)

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004854
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1493",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003599
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1523",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetObject to get the last action created for a SavedAction

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002473
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "557",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject to get the package for an Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001965

- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "600",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo on an Action to have the Server create a question that tracks the results for a Deployed Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003232
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 9 - Issue a GetObject on the package for an action to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002084
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "625",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6925-6c9d7f096a4dc99b8e34bb9d09d699963f8235f2534720facac0830700c5f8799b79fa5d2d9e828
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Deploy Action Simple Against Windows Computers

Deploy an action using the package 'Distribute Tanium Standard Utilities' to all computers that pass the filter Operating System, that contains Windows, wait for result data to be complete, and then get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005802
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013419
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "86181",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a package for inclusion in an action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001934
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "581",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to get the full object of a sensor for inclusion in a Group for an Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001778
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue an AddObject to add a list of SavedActions (6.5 logic)

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.008466
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1675",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003469
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject to get the last action created for a SavedAction

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002401

- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "560",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetObject to get the package for an Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001595
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "600",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 9 - Issue a GetResultInfo on an Action to have the Server create a question that tracks the results for a Deployed Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002773
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetObject on the package for an action to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001501
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "625",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetObject on the target_group for an action to get the full Group object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.010846
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "507",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - ID 524: Issuing an AddObject of a Question object with no Selects and the same Group used by the Action object. The number of systems that should successfully run the Action will be taken from result_info.passed_count for the Question asked when all answers for the question have reported in.

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005671
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
```

```
  "Connection": "keep-alive",
  "Content-Length": "1144",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 13 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.011850
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001651
- Step 14 Request Body
- Step 14 Response Body

- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 15 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001431
- Step 15 Request Body
- Step 15 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 16 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST

- Elapsed Time: 0:00:00.008313
- Step 16 Request Body
- Step 16 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 17 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.009190
- Step 17 Request Body
- Step 17 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1463",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 18 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003839
- Step 18 Request Body
- Step 18 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 19 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.093405
- Step 19 Request Body
- Step 19 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 20 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003132
- Step 20 Request Body
- Step 20 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1463",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 21 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003049
- Step 21 Request Body
- Step 21 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 22 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002825
- Step 22 Request Body
- Step 22 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 23 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003021
- Step 23 Request Body

- Step 23 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1463",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 24 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002721
- Step 24 Request Body
- Step 24 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 25 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002548
- Step 25 Request Body
- Step 25 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 26 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002953
- Step 26 Request Body
- Step 26 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1463",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 27 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002760
- Step 27 Request Body
- Step 27 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 28 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002531
- Step 28 Request Body
- Step 28 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "626",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 29 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002198
- Step 29 Request Body
- Step 29 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1463",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 30 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002624
- Step 30 Request Body

- Step 30 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "552",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 31 - Issue a GetResultData for an Action with the aggregate option set to False. This will return all of the Action Statuses for each computer that have run this Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002631
- Step 31 Request Body
- Step 31 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "580",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6926-7c9f86afdc2c2f45e9ecbf221adbe2421485042a7374e8aeeb4f1784b59a9d7871e3e7f03dd101ec
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Deploy Action With Params Against Windows Computers

Deploy an action using the package 'Custom Tagging - Add Tags' with parameter $1 set to 'tag_should_be_added' and parameter $2 set to 'tag_should_be_ignore' to all computers that pass the filter Operating System, that contains Windows, wait for result data to be complete, and then get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006693
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013755
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "86181",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a package for inclusion in an action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002488
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "570",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to get the full object of a sensor for inclusion in a Group for an Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001799
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
```

```
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue an AddObject to add a list of SavedActions (6.5 logic)

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.014973
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "2694",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003445
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1452",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject to get the last action created for a SavedAction

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002051
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "560",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetObject to get the package for an Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001578

- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "538",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo on an Action to have the Server create a question that tracks the results for a Deployed Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002784
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "541",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 10 - Issue a GetObject on the package for an action to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001699
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "619",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetObject on the target_group for an action to get the full Group object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.010375
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "507",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - ID 525: Issuing an AddObject of a Question object with no Selects and the same Group used by the Action object. The number of systems that should successfully run the Action will be taken from result_info.passed_count for the Question asked when all answers for the question have reported in.

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005768
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1144",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 13 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.011880
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001525
- Step 14 Request Body
- Step 14 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 15 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001990
- Step 15 Request Body
- Step 15 Response Body

- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 16 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002057
- Step 16 Request Body
- Step 16 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 17 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST

- Elapsed Time: 0:00:00.001479
- Step 17 Request Body
- Step 17 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 18 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002301
- Step 18 Request Body
- Step 18 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1445",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 19 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002625
- Step 19 Request Body
- Step 19 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "541",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 20 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002315
- Step 20 Request Body
- Step 20 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "615",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 21 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002761
- Step 21 Request Body
- Step 21 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1445",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 22 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002758
- Step 22 Request Body
- Step 22 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "541",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 23 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002489
- Step 23 Request Body
- Step 23 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "615",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 24 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002660
- Step 24 Request Body

- Step 24 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1445",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 25 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003299
- Step 25 Request Body
- Step 25 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "541",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 26 - Issue a GetResultData with the aggregate option set to True.This will return row counts of machines that have answered instead of all the data

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002618
- Step 26 Request Body
- Step 26 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "615",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 27 - Issue a GetObject for an Action in order to have access to the latest values for stopped_flag and status

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002263
- Step 27 Request Body
- Step 27 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1445",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 28 - Issue a GetResultInfo for an Action to ensure fresh data is available for a GetResultData call

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002522
- Step 28 Request Body
- Step 28 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "541",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 29 - Issue a GetResultData for an Action with the aggregate option set to False. This will return all of the Action Statuses for each computer that have run this Action

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002982
- Step 29 Request Body
- Step 29 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "569",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6927-2aa140382c7dcbeb4eab3476ec0021626f7fa2a6ac87f3ba55e7ab26add2b77a0461b7247e2ae3e0
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Action By Id

Get an action object by id

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008103
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.012553

- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6955-1ecb5e037fa01e1d9a4f45f8fbff3e57f8dba89d70dacf4bd0a59848f85391bc00f9370a08b9748
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87505",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002293
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "486",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6955-1ecb5e037fa01e1d9a4f45f8fbff3e57f8dba89d70dacf4bd0a59848f85391bc00f9370a08b9748
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Question By Id

Get a question object by id

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006206
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006814
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6956-97b438be2639ca6835344f0549d2f56bc4b585b29e59202a1eb3bc747a211ccd44d4964486c2b79
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87505",
  "content-type": "application/json"
}
```

Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004008
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "490",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6956-97b438be2639ca6835344f0549d2f56bc4b585b29e59202a1eb3bc747a211ccd44d4964486c2b79
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Saved Question By Names

Get two saved question objects by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005978
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013357
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6957-2af3ae329097dec3d362d88a9d360653716b1835f171e5ec73f8bd72dc73e8a011f7c78e8608127
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87505",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.009865
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "527",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6957-2af3ae329097dec3d362d88a9d360653716b1835f171e5ec73f8bd72dc73e8a011f7c78e8608127
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004492
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "518",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6957-2af3ae329097dec3d362d88a9d360653716b1835f171e5ec73f8bd72dc73e8a011f7c78e8608127
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Userrole By Id

Get a user role object by id.

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005994
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006953
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6958-2a673ff45a935ea342c03b91f5539bca0276815c34461c146e8b968c2c0af0b943df7e024ad1846c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87505",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001628
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6958-2a673ff45a935ea342c03b91f5539bca0276815c34461c146e8b968c2c0af0b943df7e024ad1846c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Setting By Name

Get a system setting object by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006244
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007488
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6960-11371e167410a3c3d27dcccab042b80e6663c4c924dd514fa147869596f6a38a56b9d2893c618159
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001961
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "555",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6960-11371e167410a3c3d27dcccab042b80e6663c4c924dd514fa147869596f6a38a56b9d2893c618159
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get User By Name

Get a user object by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005801
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006827
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6961-a6c385d5e9bfee16b1c86d75340cfa4f213e626e46b600eb4433723c78872c2f705c9ca3df2ed8d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001992
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6961-a6c385d5e9bfee16b1c86d75340cfa4f213e626e46b600eb4433723c78872c2f705c9ca3df2ed8d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Sensor By Id

Get a sensor object by id

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006632
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014478
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6962-806e556663cea0b7b23615b0ff964cece8c6fa24d8938d887d43ee10f8e18358ea92408976181fe
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87507",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001890
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "505",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6962-806e556663cea0b7b23615b0ff964cece8c6fa24d8938d887d43ee10f8e18358ea92408976181fe
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Sensor By Mixed

Get multiple sensor objects by id, name, and hash

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006372
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007601
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6963-f44b617a66fdcecc597a2437863b9e76d3683d54c033c4bd505ea69c8b5ffec05d18e420e4b98f91
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.040141
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "614",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6963-f44b617a66fdcecc597a2437863b9e76d3683d54c033c4bd505ea69c8b5ffec05d18e420e4b98f91
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get Whitelisted Url By Id

Get a whitelisted url object by id

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005825
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013926
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6964-55a040c3652f545b45735d684822711158d05520869fc819ff66ac741b63e509269c89f3cbd1db94
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87507",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.274445
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "480",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6964-55a040c3652f545b45735d684822711158d05520869fc819ff66ac741b63e509269c89f3cbd1db94
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Group By Name

Get a group object by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008251
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013705
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6965-3a521fa0c9310ccd02c32be07145308cf883214615912fe07587f45591f1a674c8e1474f8029d4a1
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87507",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002950
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "517",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6965-3a521fa0c9310ccd02c32be07145308cf883214615912fe07587f45591f1a674c8e1474f8029d4a1
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "941",
  "content-type": "text/xml;charset=UTF-8"
}
```


Get Sensor By Hash

Get a sensor object by hash

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005621
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006867
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6966-405a4d6d785287833998f5725a162795ddcb8e8e32b5165dafd33bfbe48c3a908ae37584235c6fa8
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87507",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.038555
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "517",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6966-405a4d6d785287833998f5725a162795ddcb8e8e32b5165dafd33bfbe48c3a908ae37584235c6fa8
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get Package By Name

Get a package object by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005193
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.012524
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6967-f398b1e06b8bbd61b08d16467a69eca7c2993423ab70b1f8445be973cd243cdc18f5dd4e80adc894
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87507",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002900
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "537",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6967-f398b1e06b8bbd61b08d16467a69eca7c2993423ab70b1f8445be973cd243cdc18f5dd4e80adc894
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Sensor By Names

Get multiple sensor objects by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006388
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006546
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6968-a86e7e1ee74b43478510f38ffd809826454e5e91dc7afac5b30e9da4e9577a094fa2531983283af
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87507",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002528
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "566",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6968-a86e7e1ee74b43478510f38ffd809826454e5e91dc7afac5b30e9da4e9577a094fa2531983283afc
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Saved Question By Name

Get saved question object by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006750
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.012952
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6969-51a3179a5e2ce7650869dbdf99edb480bd4007f47e90f2714da6a1025a5605bf0944747d5021732
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.009170
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "527",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6969-51a3179a5e2ce7650869dbdf99edb480bd4007f47e90f2714da6a1025a5605bf0944747d5021732
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get User By Id

Get a user object by id

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006342
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006269
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6970-aeb0c551b38133b56d9d3cc4f1753d9e6298940c000b442f0d93cdaafc3fda4e92d1317b4e4587c5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001448
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "482",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6970-aeb0c551b38133b56d9d3cc4f1753d9e6298940c000b442f0d93cdaafc3fda4e92d1317b4e4587c5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Sensor By Name

Get a sensor object by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005740
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.012447
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6971-a91cc2515dd0e572df8a068cef2c0e2af12b7bb513b16cf31b587ad91a8b3d49a9322cb50637487
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001938
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "521",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6971-a91cc2515dd0e572df8a068cef2c0e2af12b7bb513b16cf31b587ad91a8b3d49a9322cb50637487
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get Saved Action By Name

Get a saved action object by name

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005577
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006742
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6972-50f269071b20007ff7331b7e80b2f53e064b3ae3645d9fb08fb07332e445bd1d3ddc5052ce8a6201
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003075
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6972-50f269071b20007ff7331b7e80b2f53e064b3ae3645d9fb08fb07332e445bd1d3ddc5052ce8a6201
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get All Users

Get all user objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005774
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.006743
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6973-0cf98fbcb7fc7482bd48d0b58a7b70a0ee4dc948d97fa8ba35f4bda38397d1e2cabd2a52abdcce9
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001660
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6973-0cf98fbcb7fc7482bd48d0b58a7b70a0ee4dc948d97fa8ba35f4bda38397d1e2cabd2a52abdcce9
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Saved Actions

Get all saved action objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005721
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013732
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6974-f336dcb44b8422d72afd30cc2b9071d1cb13aa8c5f0b31a7eefc2d698d2f95ed80634b0d4793006
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87506",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.015676
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "476",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6974-f336dcb44b8422d72afd30cc2b9071d1cb13aa8c5f0b31a7eefc2d698d2f95ed80634b0d4793006
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Settings

Get all system setting objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006038
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007142
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6975-a8d699f0849961cb92259f33dc15362b54461d9ac054e2f57035be829e854b29df18a4f5f5a36bfa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87507",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004569
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "478",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6975-a8d699f0849961cb92259f33dc15362b54461d9ac054e2f57035be829e854b29df18a4f5f5a36bfa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get All Saved Questions

Get all saved question objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005370
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013595
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6976-f335d76fd02a7c82ccef7a33f9b3a7632350bc44aa32e237182c35213c777fb28910b86ce0f2cc9a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87508",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.072889
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "478",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6976-f335d76fd02a7c82ccef7a33f9b3a7632350bc44aa32e237182c35213c777fb28910b86ce0f2cc9a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Userroless

Get all user role objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007100
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013325
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6977-2b2641c688b30a79bf2deb78457fdae93098553e71f19c87666687f760e13ab5c07e27d1118efd3b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87508",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001701
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "468",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6977-2b2641c688b30a79bf2deb78457fdae93098553e71f19c87666687f760e13ab5c07e27d1118efd3b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Questions

Get all question objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005505
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007130
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6978-61e3b03c15a9ca14d1703f5e6f87356c3c51edbf0bfcdfd4196b8449918de1fcfac4540b6e65518a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87508",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.143477
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "472",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6978-61e3b03c15a9ca14d1703f5e6f87356c3c51edbf0bfcdfd4196b8449918de1fcfac4540b6e65518a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get All Groups

Get all group objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007643
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.012033
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6979-a466eac2892421e3ba6fc16249ce7d935b80f2443b7cb85369c9dcfe15a72e87da119c65e762eda4
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87508",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003675
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "469",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6979-a466eac2892421e3ba6fc16249ce7d935b80f2443b7cb85369c9dcfe15a72e87da119c65e762eda4
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Sensors

Get all sensor objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.005903
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.011725
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6980-98f0a0a83553198841a7a094980b0145afa44fe9a9fa7f7628c067edc4df738f6f54a0069d387906
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87508",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.290027
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "470",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6980-98f0a0a83553198841a7a094980b0145afa44fe9a9fa7f7628c067edc4df738f6f54a0069d387906
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Whitelisted Urls

Get all whitelisted url objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008187
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013009
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6981-c60f9ad57629fa6b39fa005d3d0dd5c2de7ee10f0a088f9290cfdeb67e9bdb682c80c5d7a89172b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87510",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.268061
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "480",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6981-c60f9ad57629fa6b39fa005d3d0dd5c2de7ee10f0a088f9290cfdeb67e9bdb682c80c5d7a89172b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Get All Clients

Get all client objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008631
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014325
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6982-93453cd0ca79099442b16bfccd073eb355105c088557acca3dd8e3b9708e812692b8a8272047fa3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87510",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001785
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "476",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6982-93453cd0ca79099442b16bfccd073eb355105c088557acca3dd8e3b9708e812692b8a8272047fa3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Packages

Get all package objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006304
- Step 1 Request Body

- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.007158
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6983-3cc1ff3c8446559d98a5700a15ce3106b11b317becff4716b35e037470c711ca51079fd83f54c80c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87510",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.007834
- Step 3 Request Body

- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "475",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6983-3cc1ff3c8446559d98a5700a15ce3106b11b317becff4716b35e037470c711ca51079fd83f54c80c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Get All Actions

Get all action objects

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007050
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014342
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6984-bb4f26ed649c869eb5b67f115eb1497f835c4631cbe1dda8cfa17b0d9167d3116690b9500840506
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87511",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find an object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.015045
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "470",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6984-bb4f26ed649c869eb5b67f115eb1497f835c4631cbe1dda8cfa17b0d9167d3116690b9500840506
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```

Ask Manual Question Simple Multiple Sensors

Ask the question 'Get Computer Name and Installed Applications from all machines', wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008149
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014113
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87512",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002321
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "565",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001886
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "574",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.007882
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "753",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 6 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013309

- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001560
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001449
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001554
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002098
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002153
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002380
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 13 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003278
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6985-df19baee504e368b3605cb68783a8097e76e834044ae23e5ae3bb5cffa9e098b0e7784fe873a3a1a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Simple Single Sensor

Ask the question 'Get Computer Name from all machines', wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007940
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json

- HTTP Method: GET
- Elapsed Time: 0:00:00.014139
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6986-5a7cd189759baf6790709943f871eef02e8ac71d33313c8ffbd9c6e55aecc814c38a584d9741721
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87511",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001766
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "565",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6986-5a7cd189759baf6790709943f871eef02e8ac71d33313c8ffbd9c6e55aecc814c38a584d9741721
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.007892
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "639",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6986-5a7cd189759baf6790709943f871eef02e8ac71d33313c8ffbd9c6e55aecc814c38a584d9741721
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013276
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6986-5a7cd189759baf6790709943f871eef02e8ac71d33313c8ffbd9c6e55aecc814c38a584d9741721
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
```

```
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001613
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6986-5a7cd189759baf6790709943f871eef02e8ac71d33313c8ffbd9c6e55aecc814c38a584d9741721
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002290
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6986-5a7cd189759baf6790709943f871eef02e8ac71d33313c8ffbd9c6e55aecc814c38a584d9741721
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001622
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6986-5a7cd189759baf6790709943f871eef02e8ac71d33313c8ffbd9c6e55aecc814c38a584d9741721
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Sensor With Parameters And Some Supplied Parameters

Ask the question 'Get Folder Name Search with RegEx Match[Program Files,Microsoft.*] from all machines', wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008065
- Step 1 Request Body
- Step 1 Response Body

- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.013871
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87511",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002496
- Step 3 Request Body
- Step 3 Response Body

- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "587",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.021312
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1003",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST

- Elapsed Time: 0:00:00.040338
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001756
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002215
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002240
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002076
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002232
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002211
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001850
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 13 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002142
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002194

- Step 14 Request Body
- Step 14 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 15 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002162
- Step 15 Request Body
- Step 15 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 16 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002877
- Step 16 Request Body
- Step 16 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 17 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005290
- Step 17 Request Body
- Step 17 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6988-05836d41c7be5addca66f76227a78b0d47fead12ac9dd8b6b2af69ce4387d294ba47baf393a4fd0e
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Multiple Sensors With Parameters And Some Supplied Parameters

Ask the question 'Get Folder Name Search with RegEx Match[Program Files, , No, No, Microsoft.*] and Computer Name from all machines', wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.016327
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014295
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87614",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002678
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "587",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002104
- Step 4 Request Body

- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "565",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.019997
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1117",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 6 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.038632
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001769
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002101
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002241
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001812
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001735
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002245
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 13 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002252
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002166
- Step 14 Request Body
- Step 14 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 15 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001882

- Step 15 Request Body
- Step 15 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 16 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002162
- Step 16 Request Body
- Step 16 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 17 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002169
- Step 17 Request Body
- Step 17 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 18 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001497
- Step 18 Request Body
- Step 18 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 19 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002106
- Step 19 Request Body
- Step 19 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 20 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002004
- Step 20 Request Body
- Step 20 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 21 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002156
- Step 21 Request Body
- Step 21 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 22 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002059
- Step 22 Request Body
- Step 22 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 23 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002309
- Step 23 Request Body
- Step 23 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 24 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002313

- Step 24 Request Body
- Step 24 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 25 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002819
- Step 25 Request Body
- Step 25 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 26 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002749
- Step 26 Request Body
- Step 26 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 27 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002601
- Step 27 Request Body
- Step 27 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 28 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002821
- Step 28 Request Body
- Step 28 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 29 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002798
- Step 29 Request Body
- Step 29 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 30 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002945
- Step 30 Request Body
- Step 30 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 31 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002904
- Step 31 Request Body
- Step 31 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 32 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002898
- Step 32 Request Body
- Step 32 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6989-0a6f5bd674c41d8e3da5b1a0fb91cefaad9e2d713b26ceb4911bd3fc82fce71420e9b4013e42581c
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Sensor With Parameters And No Supplied Parameters

Ask the question 'Get Folder Name Search with RegEx Match from all machines' using sane defaults for parameters, wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006460
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.012795
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87813",
  "content-type": "application/json"
}
```

Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002139
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "587",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013984
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "915",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.034328
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001614
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
```

```
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002247
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002092
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002088
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002095

- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002077
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001843
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 13 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001758
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002647
- Step 14 Request Body
- Step 14 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 15 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001766
- Step 15 Request Body
- Step 15 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 16 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001431
- Step 16 Request Body
- Step 16 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 17 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001358
- Step 17 Request Body
- Step 17 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 18 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001872
- Step 18 Request Body
- Step 18 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 19 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002165

- Step 19 Request Body
- Step 19 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 20 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002035
- Step 20 Request Body
- Step 20 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 21 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002250
- Step 21 Request Body
- Step 21 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 22 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002073
- Step 22 Request Body
- Step 22 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 23 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001999
- Step 23 Request Body
- Step 23 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 24 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002042
- Step 24 Request Body
- Step 24 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 25 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002114
- Step 25 Request Body
- Step 25 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 26 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002965
- Step 26 Request Body
- Step 26 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 27 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003084
- Step 27 Request Body
- Step 27 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 28 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005488

- Step 28 Request Body
- Step 28 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6991-050b6d41b647d68310f9deb32de806e13b27ac42be5cecefe7b740685b54e740d4af331c939f94b
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Sensor With Parameters And Filter

Ask the question 'Get Folder Name Search with RegEx Match[Program Files, , No, No, Microsoft.*] containing "Shared" from all machines', wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007992
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```

Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014992
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "87914",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002108
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "587",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
```

```
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.019908
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1081",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.045453
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001612
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002106
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002144
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002064
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002143
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002220

- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002236
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 13 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002221
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002199
- Step 14 Request Body
- Step 14 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 15 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002099
- Step 15 Request Body
- Step 15 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 16 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001708
- Step 16 Request Body
- Step 16 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 17 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002067
- Step 17 Request Body
- Step 17 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 18 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002111
- Step 18 Request Body
- Step 18 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 19 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002176
- Step 19 Request Body
- Step 19 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 20 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002106

- Step 20 Request Body
- Step 20 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 21 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002305
- Step 21 Request Body
- Step 21 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 22 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002164
- Step 22 Request Body
- Step 22 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 23 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002386
- Step 23 Request Body
- Step 23 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 24 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002185
- Step 24 Request Body
- Step 24 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 25 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002139
- Step 25 Request Body
- Step 25 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 26 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001818
- Step 26 Request Body
- Step 26 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 27 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002167
- Step 27 Request Body
- Step 27 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 28 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002295
- Step 28 Request Body
- Step 28 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 29 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001825

- Step 29 Request Body
- Step 29 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 30 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002186
- Step 30 Request Body
- Step 30 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 31 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002053
- Step 31 Request Body
- Step 31 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6992-c40d5e60dacfd24c809d9fa0385d0c0798b093201fc30229a8eed981c76f6dd42e081763b835f2d
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Sensor With Filter And 2 Options

Ask the question 'Get Operating System containing "Windows" from all machines' and set max_age_seconds to 3600 and value_type to 1 on the Operating System sensor, then wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.008899
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014913
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "88017",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002268
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.009065
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "784",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013911
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001928
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002026

- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002111
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f3
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 9 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001795
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6993-17fa9da998b1727bf636c12c4388c86cbd275fef1bcd8dd676fb15b62a9a60e410dca9d5d481c6f
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Sensor With Filter

Ask the question 'Get Operating System containing "Windows" from all machines', then wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006415
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```

- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.017046
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6994-82a216fcbf87d26d95d8d0dd477147367bab210c9d2aef59f5d8a31401c429700341c38ab52e8c55
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "88017",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002471
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "session": "1-6994-82a216fcbf87d26d95d8d0dd477147367bab210c9d2aef59f5d8a31401c429700341c38ab52e8c55
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.008750
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "714",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6994-82a216fcbf87d26d95d8d0dd477147367bab210c9d2aef59f5d8a31401c429700341c38ab52e8c55
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013267
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6994-82a216fcbf87d26d95d8d0dd477147367bab210c9d2aef59f5d8a31401c429700341c38ab52e8c55
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001604
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6994-82a216fcbf87d26d95d8d0dd477147367bab210c9d2aef59f5d8a31401c429700341c38ab52e8c55
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002184

- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6994-82a216fcbf87d26d95d8d0dd477147367bab210c9d2aef59f5d8a31401c429700341c38ab52e8c55
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001704
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6994-82a216fcbf87d26d95d8d0dd477147367bab210c9d2aef59f5d8a31401c429700341c38ab52e8c55
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Ask Manual Question Sensor With Parameters And Filter And Options

Ask the question 'Get Folder Name Search with RegEx Match[Program Files, , No, No, Microsoft.*] containing "Shared" from all machines' and set max_age_seconds to 3600 on the Folder Name Search with RegEx Match sensor, then wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007706
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014205
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "88017",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002391
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "587",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.019317
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1120",
  "Content-Type": "text/xml; charset=utf-8",
```

```
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.043114
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001575
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002203
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002143

- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002126
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002192
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002215
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001611
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 13 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001594
- Step 13 Request Body
- Step 13 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6995-5602dcc91eee5e83246cd8affc00dae46fea49c258488a5456894e0bc6f91aac05fb2d73881976fa
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Sensor With Filter And 3 Options

Ask the question 'Get Operating System containing "Windows" from all machines' and set max_age_seconds to 3600, all_values_flag to 1, and ignore_case_flag to 1 on the Operating System sensor, then wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006153
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET

- Elapsed Time: 0:00:00.014004
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "88119",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002142
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.009947
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "861",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 5 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013336
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 6 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001812
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002107
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002105
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001625
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6996-83e914379f96cc824a16ccdd4491c1cdd37f4144d45536b9a8c9d16d094452da2c06e64cf496ea34
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Complex Query1

Ask the question 'Get Computer Name and Folder Name Search with RegEx Match[Program Files, , No, No, Microsoft.*, test] containing "Shared" from all machines with (Operating System containing "Windows" or any Operating System not containing "Windows")' and set ignore_case_flag to 1 and or_flag to 1 on the Operating System sensors on the right hand side of the question, then wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007867
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET

- Elapsed Time: 0:00:00.013975
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "88120",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002020
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "565",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap

- HTTP Method: POST
- Elapsed Time: 0:00:00.002436
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "587",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a GetObject to get the full object of a sensor for inclusion in a Group for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001824
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 6 - Issue a GetObject to get the full object of a sensor for inclusion in a Group for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001954
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "568",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.027102
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1678",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 8 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.041812
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001588
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
```

```
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001816
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002226
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002222
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 13 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002423

- Step 13 Request Body
- Step 13 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 14 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002290
- Step 14 Request Body
- Step 14 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 15 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002092
- Step 15 Request Body
- Step 15 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6997-76265f412df09760c20cf2d988647cf481da6d8fc9a003da180e2e7675a08d5ff29d0cbdaf1e4e2a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Manual Question Complex Query2

Ask the question 'Get Computer Name and Last Logged In User and Installed Applications containing "Google (Search|Chrome)" from all machines with Installed Applications containing "Google (Search|Chrome)"' and set ignore_case_flag to 1 and or_flag to 1 on the Installed Applications sensors on the right hand side of the question, then wait for result data to be complete, and get result data

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.007472
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
```

```
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014728
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "88223",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002128
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
```

```
  "Content-Length": "565",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002414
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "571",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a GetObject to get the full object of a sensor for inclusion in a Select for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002335
- Step 5 Request Body
- Step 5 Response Body

- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "574",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 6 - Issue a GetObject to get the full object of a sensor for inclusion in a Group for a Question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002212
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "574",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue an AddObject to add a Question object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST

- Elapsed Time: 0:00:00.012807
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "1174",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "769",
  "content-type": "text/xml;charset=UTF-8"
}
```


Step 8 - Issue a GetObject on the recently added object in order to get the full object

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.012531
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "494",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001716
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001635
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002136
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 12 - Issue a GetResultData to get answers for a question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001741
- Step 12 Request Body
- Step 12 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-6998-ec9c0cc41fcb67257e819c5c8f1ff7f7dc5028d8ee80f6d3ccc0d53f196486a1ed62c4b1f4f045a5
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Saved Question Refresh Data

Get the Saved Question object for Installed Applications, ask the server to refresh the data vailable, wait for the new question spawned to complete results, then get the latest result data available for that Saved Question

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006322
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json
- HTTP Method: GET
- Elapsed Time: 0:00:00.014431

- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "88325",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find saved question objects

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.013249
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "527",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 4 - Issue a GetObject to get the full object of the last question asked by a saved question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST

- Elapsed Time: 0:00:00.003820
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "21616",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a GetResultInfo for a saved question in order to issue a new question, which refreshes the data for that saved question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.006532
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "542",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 6 - Issue a GetObject for the saved question in order get the ID of the newly asked question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.005835
- Step 6 Request Body
- Step 6 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "538",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 7 - Issue a GetObject to get the full object of the last question asked by a saved question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002829
- Step 7 Request Body
- Step 7 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "942",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 8 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.001634
- Step 8 Request Body
- Step 8 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 9 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002004
- Step 9 Request Body
- Step 9 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
```

```
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 10 - Issue a GetResultInfo for a Question to check the current progress of answers

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.002525
- Step 10 Request Body
- Step 10 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "498",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 11 - Issue a GetResultData to get the answers for the last asked question of this saved question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004172
- Step 11 Request Body
- Step 11 Response Body
- Request Headers:

```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7000-0b3a77745c282aea1f47fc9b35ea6d94e18e7f95336eb79657a3b449756d4ce45f272fff7ec4e30a
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Ask Saved Question By Name

Get the Saved Question object for Installed Applications then get the latest result data available for that Saved Question

Step 1 - Authenticate to the SOAP API via /auth

- URL: https://10.0.1.240:443/auth
- HTTP Method: GET
- Elapsed Time: 0:00:00.006143
- Step 1 Request Body
- Step 1 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "password": "VGFuaXVtMjAxNSE=",
  "username": "QWRtaW5pc3RyYXRvcg=="
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "135",
  "content-type": "text/plain; charset=us-ascii"
}
```


Step 2 - Get the server version via /info.json

- URL: https://10.0.1.240:443/info.json

- HTTP Method: GET
- Elapsed Time: 0:00:00.013685
- Step 2 Request Body
- Step 2 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip, deflate",
  "Connection": "keep-alive",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7001-95ad8b27d21e8c51a44e4bb4f8e125350dd62c00fb17dbb01569a4b251a5702950f46c77201f7432
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-length": "88324",
  "content-type": "application/json"
}
```


Step 3 - Issue a GetObject to find saved question objects

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.012031
- Step 3 Request Body
- Step 3 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "527",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7001-95ad8b27d21e8c51a44e4bb4f8e125350dd62c00fb17dbb01569a4b251a5702950f46c77201f7432
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

Step 4 - Issue a GetObject to get the full object of the last question asked by a saved question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.003569
- Step 4 Request Body
- Step 4 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "21616",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7001-95ad8b27d21e8c51a44e4bb4f8e125350dd62c00fb17dbb01569a4b251a5702950f46c77201f7432
}
```


- Response Headers:


```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```


Step 5 - Issue a GetResultData to get the answers for the last asked question of this saved question

- URL: https://10.0.1.240:443/soap
- HTTP Method: POST
- Elapsed Time: 0:00:00.004121
- Step 5 Request Body
- Step 5 Response Body
- Request Headers:


```
{
  "Accept": "*/*",
  "Accept-Encoding": "gzip",
  "Connection": "keep-alive",
  "Content-Length": "526",
  "Content-Type": "text/xml; charset=utf-8",
  "User-Agent": "python-requests/2.6.0 CPython/2.7.10 Darwin/14.5.0",
  "session": "1-7001-95ad8b27d21e8c51a44e4bb4f8e125350dd62c00fb17dbb01569a4b251a5702950f46c77201f7432
}
```


- Response Headers:

```
{
  "connection": "keep-alive",
  "content-encoding": "gzip",
  "content-type": "text/xml;charset=UTF-8",
  "transfer-encoding": "chunked"
}
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

d

p

r

t

X

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X