# Risksense API Library

***Release 2.0***

**arockia,thahasina,burr**

**Feb 08, 2023**

# CONTENTS

Welcome to the risksense package. Get started in understanding the different api functionalities in the risksense platform along with the tools to get you started.

CHAPTER

ONE

# BROWSE THE REPOSITORY ON GITHUB

The master branch will always be the most recent stable version.

## 1.1 Installation

### 1.1.1 Install Python

Download and install a *supported version* of python for your platform.

### 1.1.2 Download the entire repository using git

This is really only for those who wish to contribute to this project, although you can clone the repository using the below command.

```
$ git clone git://github.com/risksense/risksense_tools.git
```

### 1.1.3 Browse the repository on GitHub

The master branch will always be the most recent stable version.

### 1.1.4 Dependency module

Do make sure you download the dependency modules by running the requirements.txt using the command below

```
$ pip install -r requirements.txt
```

## 1.2 Using Risksense API Library

To begin make sure you provide the system path to the lib package before importing the script example.

```
>>> sys.path.insert(0, os.path.join(os.path.dirname(os.path.dirname(os.path.dirname(os.
↪path.abspath(__file__)))), 'lib'))
```

To use risksense lib package please ensure you import risksense api in your script

```
>>> import risksense_api as rsapi
```

To perform usage of the subject functions you must first create an object and use that object for subject function definitions. Please ensure you should provide the client id either during function definitions or by setting a default client id using the below function `set_default_client_id()`

```
>>> self.rs=rs_api.RiskSenseApi(self._rs_platform_url, api_key)
>>> self.rs.set_default_client_id(self._client_id)
```

where self._rs_platform_url is the url of the platform and apikey is the user apikey

Now post the risksense object creation, you can use the object `self.rs` for using functions in risksense api packages

```
>>> self.rs.{subjectname}.{functionname}
where
    subjectname - The subject module present in the lib package
    functionname - The functionname define for  that particular subject
```

### 1.2.1 Applications (risksense_api.__subject.__applications.__applications)

****Application module defined for different application related api endpoints.****

**class** risksense_api.__subject.__applications.__applications.**Applications**(*profile*)

Bases: Subject

**Class for Applications function defintions**.

To utlise Applications function:

**Parameters**
**profile** – Profile Object

**Usage:**
`self.{risksenseobjectname}.applications.{function}`

---

**Examples**

To delete application using *delete()* function

```
>>> self.{risksenseobject}.applications.delete({applicationfilter})
```

---

**__init__**(*profile*)

**Initialization of Applications Object** .

**Parameters**
**profile** – Profile Object

**create**(*name*, *groupids*, *networkid*, *applicationurl*, *criticality*, *externality=False*, *csvdump=False*, *client_id=None*)

Create an application

**Parameters**

- **name** (str) – Name of the application .

- **groupids** (list) – ids of the groups you want it to be assigned to
- **networkid** (int) – network id. Id of network the application to be a part of
- **applicationurl** (str) – url of the application.
- **criticality** (int) – the application criticality
- **externality** (bool) – the application whether external or internal. Externality is true if application is external , false if internal
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response.

---

**Examples**

To create an application:

```
>>> self.[risksenseobject].applications.create('applicationname',[1,2,3],123,
↪'webpagetest.org',5,False)
```

---

**Note:** You can also dump the job id of the created application using csvdump=True argument:

```
>>> self.[risksenseobject].applications.create('applicationname',[1,2,3],123,
↪'webpagetest.org',5,False,csvdump=True)
```

---

**delete**(*filterrequest*, *csvdump=False*, *client_id=None*)

Deletes an application

**Parameters**

- **filterrequest** (list) – Search filters .
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The Jsonified response.

---

**Examples**

To delete an application:

```
>>> self.[risksenseobject].applications.delete([])
```

**Note:** You can also dump the application data that is going to be deleted by adding a `csvdump=True` argument:

```
>>> self.[risksenseobject].applications.delete([],csvdump=True)
```

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

**Exports and Downloads a file based on the filters defined** .

**Parameters**

- **filename** (str) – Name of the file to export as
- **filters** (dict) – Application search filters based on which the export performs
- **client_id** (typing.Optional[int]) – The client id to get the data from. If not supplied, takes default client id

*IGNORE Internal function*

**Examples**

```
>>> self.[risksenseobject].applications.downloadfilterinexport('applicationdata
↪',[])
```

**list_application_filter_fields**(*client_id=None*)

Lists application filter fields.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The JSON output from the platform is returned, listing the available filters.

**Example**

```
>>> self.[risksenseobject].applications.list_application_filter_fields()
```

**get_single_search_page**(*search_filters*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns applications based on the provided filter(s) and other parameters for a single page.

**Parameters**

- **search_filters** (list) – List of dictionaries containing filter parameters.
- **page_num** (int) – Page number of results to be returned.
- **page_size** (int) – Number of results to be returned per page.
- **sort_field** (str) – Name of field to sort results on.

- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The paginated JSON response from the platform is returned.

---

**Example**

An example to get single search page of applications data

```
>>> self.{risksenseobject}.applications.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as 'change page_size` or *page_num* etc.

```
>>> self.{risksenseobject}.applications([],page_num=2,page_size=10)
```

---

**get_groupby_application**(*client_id=None*)

Get groupby keymetrics for applications

**Parameters**
**client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
dict

**Returns**
The keymetrics for groupby

---

**Example**

```
>>>  self.{risksenseobject}.applications.get_groupby_application()
```

---

**IGNORE INTERNAL FUNCTION** .. note:: This function just returns the groupby key metrics

**groupby_application**(*filters=[]*, *sortorder=None*, *csvdump=False*, *client_id=None*)

Get groupby values for applications

**Parameters**

- **filters** (list) – The filters which will populate in groupby
- **sortorder** (typing.Optional[str]) – The order to sort the groupby values, please choose ASC for ascending and DESC for descending
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[bool]) – The client id , if none, default client id is taken

**Return type**
dict

**Returns**
The groupby values of the application.

---

**Example**

```
>>> self.{risksenseobject}.applications.groupby_application({filter})
```

The filter must be provided for the group by to be used. The groupby fields will be displayed in the *terminal* and you must choose a *group by* filter to which the data will be populated

---

**Note:** This function also has an option to dump the data in a csv by a simple argument, `csvdump=True`

```
>>> self.{risksenseobject}.applications.groupby_application({filter},
↪csvdump=True)
```

---

**search**(*search_filters*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns applications based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **page_size** (`int`) – The number of results per page to be returned.
- **sort_field** (`str`) – The field to be used for sorting results returned.
- **sort_dir** (`str`) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`list`

**Returns**

A list containing all applications returned by the search using the filter provided.

**Example:**
An example to search for application data is

```
>>> self.{risksenseobject}.applications.search([])
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.search([],csvdump=True)
```

---

**get_count**(*search_filters*, *client_id=None*)

Gets a count of applications identified using the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The number of applications identified using the filter(s).

---

**Example**

To get count of the appplications

```
>>> self.[risksenseobject].applications.get_count([])
```

---

**merge_application**(*searchfilters*, *application_id_to_merge_to*, *csvdump=False*, *client_id=None*)

Merges applications based on search filters to the application id provided.

**Parameters**

- **searchfilters** (list) – A list of dictionaries containing filter parameters.
- **application_id_to_merge_to** (int) – Application id to merge to.
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

An example to use merge_application is

```
>>> self.[risksenseobject].applications.merge_application([],123)
```

---

**Note:** You can also dump the applications that are going to be merged before merging them by csvdump=True argument

```
>>> self.[risksenseobject].applications.merge_application([],123,csvdump=True)
```

---

**set_asset_criticality**(*filter*, *assetcriticality*, *csvdump=False*, *client_id=None*)

Set asset criticality for the application.

**Parameters**

- **filter** (list) – A list of dictionaries containing filter parameters.

- **assetcriticality** (int) – The asset criticality to set the filter specified applications to.
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

To set asset criticality based on id

```
>>> self.{risksenseobject}.applications.set_asset_criticality([{"field":"id",
↪"exclusive":False,"operator":"EXACT","orWithPrevious":False,"implicitFilters
↪":[],"value":"1234"}],3)
```

---

**Note:** You can also dump the applications to which asset criticality should be changed by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.set_asset_criticality([{"field":"id",
↪"exclusive":False,"operator":"EXACT","orWithPrevious":False,"implicitFilters
↪":[],"value":"1234"}],3,csvdump=True)
```

---

**set_address_type**(*filter*, *addresstype*, *csvdump=False*, *client_id=None*)

Set address type for the application.

**Parameters**

- **filter** (list) – A list of dictionaries containing filter parameters.
- **addresstype** (str) – The address type whether external or internal, provide string external for external and internal for internal
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

To set address type based on id

```
>>> self.{risksenseobject}.applications.set_address_type([{"field":"id",
↪"exclusive":False,"operator":"EXACT","orWithPrevious":False,"implicitFilters
↪":[],"value":"1234"}],"EXTERNAL")
```

---

**Note:** You can also dump the applications which the address type will be set by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.set_address_type([{"field":"id",
↪"exclusive":False,"operator":"EXACT","orWithPrevious":False,"implicitFilters
↪":[],"value":"1234"}],"EXTERNAL",csvdump=True)
```

---

**edit_application**(*filter*, *name*, *url*, *csvdump=True*, *client_id=None*)

Edit an application.

**Parameters**

- **filter** (`list`) – A list of dictionaries containing filter parameters.
- **name** (`str`) – Name of the application
- **url** (`str`) – Url of the application
- **csvdump** (`bool`) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

To edit an application based on an id `1234` from platform

```
>>> self.{risksenseobject}.applications.edit_application([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],name='test1',url='10.1.1.1/
↪app')
```

---

**Note:** You can also dump the applications which are edited by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.edit_application([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],name='test1',url='10.1.1.1/
↪app',csvdump=True)
```

---

**add_tag**(*search_filters*, *tag_id*, *csvdump=False*, *client_id=None*)

Add a tag to application(s).

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.

- **tag_id** (int) – The tag ID to add to the application(s).
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

To add a tag for an application id 1234 to tag 123

```
>>> self.rs.applications.add_tag([{"field":"id","exclusive":False,"operator":"IN
↪","value":"1234"}],123)
```

---

**Note:** You can also dump the applications to which tags are added post tag addition by csvdump=True argument

```
>>> self.rs.applications.add_tag([{"field":"id","exclusive":False,"operator":"IN
↪","value":"1234"}],123,csvdump=True)
```

---

**remove_tag**(*search_filters*, *tag_id*, *csvdump=False*, *client_id=None*)

Remove a tag from application(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – The tag ID to remove from the application(s).
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

To remove a tag from an application id 1234 from tag 123

```
>>> self.rs.applications.remove_tag([{"field":"id","exclusive":False,"operator":
↪"IN","value":"1234"}],123)
```

---

**Note:** You can also dump the applications to which tags are removed before tag removal by `csvdump=True` argument

```
>>> self.rs.applications.remove_tag([{"field":"id","exclusive":False,"operator":
↪"IN","value":"1234"}],123,csvdump=True)
```

**getexporttemplate**(*client_id=None*)

Gets configurable export template for Applications.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.applications.getexporttemplate()
```

This gets all the export templates for applications

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*)

Initiates an export job on the platform for application(s) based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Available options ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform from is returned.

**Example**

An example to use export is

```
>>> self.{risksenseobject}.applications.export([],'testingexport')
```

You can change the filetype to any of the names above or even the other positional arguments as mentioned

```
>>> self.{risksenseobject}.applications.export([],'testingexport',
↪filetype=ExportFileType.JSON)
```

---

**network_move**(*search_filters*, *network_id*, *force_merge=False*, *csvdump=False*, *client_id=None*)

Move an application to a different network.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **network_id** (`int`) – The ID of the network the application should be moved to.
- **force_merge** (`bool`) – Boolean indicating whether or not a merge should be forced.
- **csvdump** (`bool`) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

To move an application in id `123` to network `1234`

```
>>> self.{risksenseobject}.applications.network_move([{"field":"id","exclusive
↪":False,"operator":"EXACT","orWithPrevious":False,"implicitFilters":[],"value
↪":"123"}],1234)
```

---

**Note:** You can also dump the applications that are being moved by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.network_move([{"field":"id","exclusive
↪":False,"operator":"EXACT","orWithPrevious":False,"implicitFilters":[],"value
↪":"123"}],1234,csvdump=True)
```

---

**run_urba**(*search_filters*, *csvdump=False*, *client_id=None*)

Initiates the update of remediation by assessment for application(s) specified in filter(s).

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **csvdump** (`bool`) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

An example to use run_urba for an application 123 is

```
>>> self.{risksenseobject}.applications.run_urba([{"field":"id","exclusive
↪":False,"operator":"IN","value":"123"}])
```

---

**Note:** You can also dump the applications to which urba is being run by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.run_urba([{"field":"id","exclusive
↪":False,"operator":"IN","value":"123"}],csvdump=True)
```

---

**add_note**(*search_filters*, *note*, *csvdump=False*, *client_id=None*)

Add a note to applications based on search filters.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **note** (`str`) – A note to be added to the application(s).
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

An example to use add_note is

```
>>> self.{risksenseobject}.applications.add_note([],'test')
```

---

**Note:** You can also dump the applications to which notes will be added post adding the note by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.add_note([],'test',csvdump=True)
```

---

**get_model**(*client_id=None*)

Get available projections and models for Applications.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Application projections and models are returned.

---

**Example**

An example to use get_model is

```
>>> self.{risksenseobject}.applications.get_model()
```

---

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
Value suggestions

---

**Example**

An example to use suggest for assessment labels is

```
>>> self.{risksenseobject}.applications.suggest([],{"field":"assessment_labels",
↪"exclusive":False,"operator":"WILDCARD","value":"","implicitFilters":[]})
```

---

**add_group**(*search_filter*, *group_ids*, *csvdump=False*, *client_id=None*)

Add application(s) to one or more groups.

**Parameters**

- **search_filter** (list) – Search filter
- **group_ids** (list) – List of Group IDs to add to application(s).
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
int

**Returns**
Job ID of group add job

**Example**

An example to use add_group is

```
>>> self.{risksenseobject}.applications.add_group([],[2,3,4])
```

**Note:** You can also dump the applications which will be addedd to the groups by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.add_group([],[2,3,4],csvdump=True)
```

**remove_group**(*search_filter*, *group_ids*, *csvdump=False*, *client_id=None*)

Remove application(s) from one or more groups.

**Parameters**

- **search_filter** (`list`) – Search filter
- **group_ids** (`list`) – List of Group IDs to add to application(s).
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID

**Return type**
`int`

**Returns**
Job ID of group remove job

**Example**

An example to use remove_group is

```
>>> self.{risksenseobject}.applications.remove_group([],[2,3,4])
```

**Note:** You can also dump the applications which will be removed from the groups by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.remove_group([],[2,3,4],csvdump=True)
```

**apply_system_filters**(*csvdump=False*, *client_id=None*)

Get data from system filters for applications.

**Parameters**

- **csvdump** (`bool`) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`list`

**Returns**
The data of the application findings based on the system filter chosen.

---

**Example**

An example to use apply_system_filters is

```
>>> self.{risksenseobject}.applications.apply_system_filters()
```

The system filters will be displayed in the terminal to which you must provide a key value and the data returned will reflect based on the system filter chosrn

---

**Note:** You can also dump the applications from the system filters search by `csvdump=True` argument

```
>>> self.{risksenseobject}.applications.apply_system_filters(csvdump=True)
```

---

### 1.2.2 Application Findings (risksense_api.__subject.__application_findings.__application_findings)

**Application Findings module defined for different application findings related api endpoints.**

**class** risksense_api.__subject.__application_findings.__application_findings.**ApplicationFindings**(*profile*)

Bases: `Subject`

**Class for Application Findings function defintions**.

To utlise Application Findings function:

**Parameters**
**profile** (`object`) – Profile Object

**Usage:**
`self.{risksenseobjectname}.application_findings.{function}`

---

**Examples**

To get model for application findings using *get_model()* function

```
>>> self.{risksenseobject}.application_findings.get_model()
```

---

**__init__**(*profile*)

**Initialization of Application Findings Object** .

**Parameters**
**profile** (`object`) – Profile Object

**create**(*applicationids*, *assessmentid*, *synopsis*, *description*, *severity*, *sourceid*, *scanneruuid*, *title*, *solution*, *parameter*, *payload*, *request*, *response*, *filterrequests*, *cweids*, *applicationurl*, *isSelectedAll=False*, *csvdump=False*, *client_id=None*)

Creates application finding.

**Parameters**

- **applicationids** (list) – A list containing application ids the findings are part of
- **assessmentid** (int) – Assessment id of the finding
- **synopsis** (str) – Synopsis for the application finding
- **description** (str) – description for the application finding
- **severity** (str) – Application severity
- **sourceid** (str) – Sourceid of the application
- **scanneruuid** (str) – scanneruuid of the application
- **title** (str) – title for the application
- **solution** (str) – solution for the application
- **parameter** (str) – parameter for the application
- **payload** (str) – payload for the application
- **request** (str) – request for the application
- **response** (str) – request for the application
- **filterrequests** (list) – filterrequests for the application as a list
- **cweids** (list) – cwe ids for the application as a list
- **applicationurl** (str) – applicationurl for the application as a list
- **isSelectedAll** (bool) – whether isselectedall
- **csvdump** (bool) – dumps id to csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
<module 'json' from '/home/docs/.pyenv/versions/3.7.9/lib/python3.7/json/__init__.py'>

**Returns**
Jsonified response.

---

**Example**

To create an application finding

```
>>> self.{risksenseobject}.application_findings.create(123, 123, "Application/
↪Environment information being disclosed", "[p]Any part of the application␣
↪which reveals details", "8", "public_key_pinning", "aaaabbbb-cccc-ddd",
↪"vulnerability test 4", "[p]Applications should not reveal any debug or", "", "",
↪"", "", [{"field": "id", "exclusive": False, "operator": "IN", "value": ""}], 1, 2, "/
↪webpage.html")
```

---

**Note:** You can also dump the application finding job id created in a csv using `csvdump=True`:

```
>>> self.{risksenseobject}.application_findings.create(123, 123, "Application/
↪Environment information being disclosed", "[p]Any part of the application␣
↪which reveals details", "8", "public_key_pinning", "aaaabbbb-cccc-ddd",
```

(continues on next page)

(continued from previous page)

```
↪"vulnerability test 4","[p]Applications should not reveal any debug or","","",
↪"","",[{"field":"id","exclusive":False,"operator":"IN","value":""}],[1,2,],"/
↪webpage.html",csvdump=True)
```

---

**update**(*applicationfindingid*, *applicationids*, *assessmentid*, *applicationurl*, *severity*, *sourceid*, *title*, *description*, *solution*, *synopsis*, *notes*, *cweids*, *request*, *response*, *parameter*, *payload*, *vulnrequestid*, *csvdump=False*, *client_id=None*)

Update application finding.

**Parameters**

- **applicationfindingid** (int) – An id of the application finding to update
- **applicationids** (int) – A list containing application ids the findings are part of
- **assessmentid** (int) – Assessment id of the finding
- **applicationurl** (str) – Url of the application
- **severity** (int) – Application severity
- **sourceid** (int) – Sourceid of the application
- **title** (str) – title for the application
- **description** (str) – description for the application finding
- **solution** (str) – solution for the application
- **synopsis** (str) – Synopsis for the application finding
- **notes** (str) – notes for application finding
- **cweids** (int) – cwe id
- **request** (str) – request for the application
- **response** (str) – request for the application
- **parameter** (str) – parameter for the application
- **payload** (str) – payload for the application
- **vulnrequestid** (int) – payload for the application
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response.

---

**Example**

To Update the application findings

```
>>> self.[risksenseobject].application_findings.
↪update(applicationfindingid=3205739,applicationids=11519,assessmentid=107556,
↪applicationurl="/webpage.html",severity=9,sourceid=1234,title='vulnerability␣
↪test 8',description="[p]Any part of the application which reveals details",
↪solution='something to work on etc',synopsis='something to work on etc',notes=
↪'something to work on etc',cweids=79,request='',response='',parameter='',
↪payload='',vulnrequestid=1234)
```

**Note:** You can also dump the job id of the application findings using `csvdump=True` argument:

```
>>> self.[risksenseobject].application_findings.
↪update(applicationfindingid=3205739,applicationids=11519,assessmentid=107556,
↪applicationurl="/webpage.html",severity=9,sourceid=1234,title='vulnerability␣
↪test 8',description="[p]Any part of the application which reveals details",
↪solution='something to work on etc',synopsis='something to work on etc',notes=
↪'something to work on etc',cweids=79,request='',response='',parameter='',
↪payload='',vulnrequestid=1234,csvdump=True)
```

`get_model`(*client_id=None*)

Get available projections and models for application findings.

**Parameters**
`client_id` (`typing.Optional[int]`) – Client ID

**Return type**
`dict`

**Returns**
Application findings projections and models are returned.

**Example**

An example to use get_model is

```
>>> self.[risksenseobject].application_findings.get_model()
```

`list_applicationfinding_filter_fields`(*client_id=None*)

Lists application finding filter fields.

**Parameters**
`client_id` (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`list`

**Returns**
The JSON output from the platform is returned, listing the available filter fields.

**Examples**

```
>>> self.{risksenseobject}.application_findings.list_applicationfinding_filter_
↪fields()
```

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
Value suggestions

**Example**

To use suggest function is

```
>>> self.{risksenseobject}.application_findings.suggest([],{})
```

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns application findings based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.

**Return type**
list

**Returns**
A list containing all application findings returned by the search using the filter provided.

**Example**

An example to search for application finding data is

```
>>> self.{risksenseobject}.application_findings.search([])
```

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.search([],csvdump=True)
```

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

**Exports and Downloads a file based on the filters defined** .

**Parameters**

- **filename** (str) – Name of the file to export as
- **filters** (list) – Application findings search filters based on which the export performs
- **client_id** (typing.Optional[int]) – The client id to get the data from. If not supplied, takes default client id

**IGNORE INTERNAL FUNCTION*

**Examples**

```
>>>  self.{risksenseobject}.application_findings.downloadfilterinexport(
↪'applicationfindingsdata',[])
```

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns application findings based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – The projection to use for API call. Projection.BASIC or Projection.DETAIL
- **page_num** (int) – Page number of results to be returned.
- **page_size** (int) – Number of results to be returned per page.
- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The paginated JSON response from the platform is returned.

**Example**

An example to get single search page of application findings data

```
>>> self.[risksenseobject].application_findings.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.[risksenseobject].application_findings.get_single_search_page([],page_
↪num=2,page_size=10)
```

**apply_system_filters**(*csvdump=False*, *client_id=None*)

Get data from system filters for application findings.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, it will use the profile's default Client ID.
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if it's not needed.

**Return type**
list

**Returns**
The data of the system filter based application findings values are returned

**Example**

An example to use apply_system_filters is

```
>>> self.[risksenseobject].application_findings.apply_system_filters()
```

The system filters will be displayed in the terminal to which you must provide a key value and the data returned will reflect based on the system filter chosen

**Note:** You can also dump the application findings from the system filters search by csvdump=True argument

```
>>> self.[risksenseobject].application_findings.apply_system_
↪filters(csvdump=True)
```

**get_groupby_appfinding**(*client_id=None*)

Gets all the groupby key metrics for application findings

**Parameters**
**client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
dict

**Returns**
The group by key metrics

**IGNORE INTERNAL FUNCTION* .. admonition:: Example

```
>>>  self.{risksenseobject}.application_findings.get_groupby_appfinding()
```

---

**Note:** This function just returns the groupbyfields

---

**groupby_appfinding**(*filters=[]*, *sortorder=None*, *csvdump=False*, *client_id=None*)

Gets groupby data for all application finding

**Parameters**

- **filters** (list) – The filters which will be used for groupby
- **sortorder** (typing.Optional[str]) – The order to sort the groupby values, please choose ASC for ascending and DESC for descending
- **csvdump** (bool) – Whether to export the data populated, if false will not export
- **client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
dict

**Returns**
Jsonified response

---

**Example**

```
>>>  self.{risksenseobject}.application_findings.groupby_appfinding({filter})
```

The filter must be provided for the group by to be used. The groupby fields will be displayed in the *terminal* and you must choose a *group by* filter to which the data will be populated

---

**Note:** This function also has an option to dump the data in a csv by a simple argument, `csvdump=True`

```
>>>  self.{risksenseobject}.application_findings.groupby_appfinding({filter},
↪csvdump=True)
```

---

**get_count**(*search_filters*, *client_id=None*)

Gets a count of application findings identified using the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The number of application findings identified using the provided filter(s).

**Example**

An example to use get count function is as follows

```
>>> self.{risksenseobject}.application_findings.get_count([])
```

**add_tag**(*search_filters*, *tag_id*, *csvdump=False*, *client_id=None*)

Add a tag to application finding(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – ID of tag to tbe added to application findings(s).
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

**Example**

An example to add a tag is

```
>>> self.{risksenseobject}.application_findings.add_tag([],1234)
```

**Note:** You can also dump the application findings from the search filters post the tag completion for more information by csvdump=True argument

```
>>> self.{risksenseobject}.application_findings.add_tag([],1234,csvdump=True)
```

**remove_tag**(*search_filters*, *tag_id*, *csvdump=False*, *client_id=None*)

Remove a tag to application finding(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – ID of tag to be removed from application findings(s).
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, it will use the profile's default Client ID.
- **csvdump** (bool) – dumps the data in csv

**Return type**
int

**Returns**
The job ID is returned.

**Example**

An example to use remove tag is

```
>>> self.{risksenseobject}.application_findings.remove_tag([],123)
```

**Note:** You can also dump the application findings which the tags will be removed from with a `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.remove_tag([],123,csvdump=True)
```

**assign**(*search_filters*, *user_ids*, *csvdump=False*, *client_id=None*)

Assign user(s) to application findings.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **user_ids** (`list`) – A list of user IDs.
- **csvdump** (`bool`) – Dumps data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID in the platform is returned.

**Example**

**Lets assign user *123* to application findings based on filter of patch id *123***

```
>>> self.{risksenseobject}.application_findings.assign([{"field":"source_
↪patch_ids","exclusive":False,"operator":"IN","value":"123"}],[123])
```

**Note:** You can also dump the application findings data before assigning them to users using `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.assign([{"field":"source_patch_
↪ids","exclusive":False,"operator":"IN","value":"123"}],[123],csvdump=True)
```

**unassign**(*search_filters*, *user_ids*, *csvdump=False*, *client_id=None*)

Unassigns user(s) from application findings.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **user_ids** (`list`) – A list of user IDs to which application findings to be unassigned.

- **csvdump** (bool) – Dumps data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform is returned.

---

**Example**

Lets unassign user *123* from application findings based on filter of patch id *123*

```
>>> self.{risksenseobject}.application_findings.unassign([{"field":"source_
↪patch_ids","exclusive":False,"operator":"IN","value":"123"}],[123])
```

---

**Note:** You can also dump the application findings data before unassigning them from users using `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.unassign([{"field":"source_
↪patch_ids","exclusive":False,"operator":"IN","value":"123"}],[123],
↪csvdump=True)
```

---

**getdynamiccolumns**(*client_id=None*)

Gets Dynamic columns for the application findings.

**Parameters**
**client_id** (typing.Optional[int]) – If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Dynamic columns

---

**Examples**

```
>>> self.{risksenseobject}.application_findings.getdynamiccolumns()
```

---

**getexporttemplate**(*client_id=None*)

Gets configurable export template for application findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.application_findings.getexporttemplate()
```

This gets all the export templates for application findings

**getexporttemplatebyid**(*export_id=None*, *client_id=None*)

Gets configurable export template by id for application findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.application_findings.getexporttemplatebyid(id)
```

This gets the export template of the id for application findings

**Parameters**
**export_id** (typing.Optional[int]) –

**getexporttemplates**(*client_id=None*)

Gets created existing export template for application findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

**Example**

An example to use getexporttemplates

```
>>> self.{risksenseobject}.application_findings.getexporttemplates()
```

This gets all the export templates for application findings

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *export_id=None*, *client_id=None*)

Initiates an export job on the platform for application finding(s) based on the provided filter(s), by default fetches all the columns data.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Available options ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **export_id** (typing.Optional[int]) – The export id of an existing export template to use

**Return type**
int

**Returns**
The job ID in the platform from is returned.

---

**Example**

An example to use export is

```
>>> self.[risksenseobject].application_findings.export([],'testingexport')
```

You can change the filetype to any of the names above or even the other positional arguments as mentioned

```
>>> self.[risksenseobject].application_findings.export([],'testingexport',file_
↪type=ExportFileType.JSON)
```

---

**subscribe_new_open_ransomware_findings**(*client_id=None*)

Subscribes the user to new open ransomware findings

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use subscribe_new_open_ransomware_findings()

```
>>> self.[risksenseobject].application_findings.subscribe_new_open_ransomware_
↪findings()
```

This helps the user subscrive to new open ransomware findings

---

**unsubscribe_new_open_ransomware_findings**(*client_id=None*)

Unsubscribes the user from new open ransomware findings

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the unsubscription that was performed

---

**Example**

An example to use unsubscribe_new_open_ransomware_findings()

```
>>> self.{risksenseobject}.application_findings.unsubscribe_new_open_ransomware_
↪findings()
```

This helps the user unsubscribe from new open ransomware findings

---

**subscribe_new_open_critical_findings_vrr**(*client_id=None*)

Subscribes the user to new open critical findings based on vrr

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use subscribe_new_open_critical_findings_vrr()

```
>>> self.{risksenseobject}.application_findings.subscribe_new_open_critical_
↪findings_vrr()
```

This helps the user subscribe to new open critical findings based on vrr.

---

**unsubscribe_new_open_critical_findings_vrr**(*client_id=None*)

Unsubscribes the user from new open critical findings

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use unsubscribe_new_open_critical_findings_vrr()

```
>>> self.{risksenseobject}.application_findings.unsubscribe_new_open_critical_
↪findings_vrr()
```

This helps the user to unsubscribe from new open critical findings based on vrr.

---

**subscribe_new_open_critical_findings_severity**(*client_id=None*)

Subscribes the user to new open critical findings based on severity

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use subscribe_new_open_critical_findings_severity()

```
>>> self.{risksenseobject}.application_findings.subscribe_new_open_critical_
↪findings_severity()
```

This helps the user subscribe to new open critical findings based on severity.

---

**unsubscribe_new_open_critical_findings_severity**(*client_id=None*)

Unsubscribes the user from new open critical findings based on severity

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use unsubscribe_new_open_critical_findings_severity()

```
>>> self.{risksenseobject}.application_findings.unsubscribe_new_open_critical_
↪findings_severity()
```

This helps the user unsubscribe from new open critical findings based on severity.

---

**subscribe_new_open_high_findings_vrr**(*client_id=None*)

Subscribes the user to new open high findings based on vrr

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use subscribe_new_open_high_findings_vrr()

```
>>> self.{risksenseobject}.application_findings.subscribe_new_open_high_
↪findings_vrr()
```

This helps the user subscribe to new open high findings based on vrr.

---

**unsubscribe_new_open_high_findings_vrr**(*client_id=None*)

Unsubscribe the user from new open high findings based on vrr

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use unsubscribe_new_open_high_findings_vrr()

```
>>> self.{risksenseobject}.application_findings.unsubscribe_new_open_high_
↪findings_vrr()
```

This helps the user unsubscribe from new open high findings based on vrr.

---

**subscribe_new_open_high_findings_severity**(*client_id=None*)

Subscribes the user to new open high findings based on severity

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

**Example**

An example to use subscribe_new_open_high_findings_severity()

```
>>> self.[risksenseobject].application_findings.subscribe_new_open_high_
↪findings_severity()
```

This helps the user subscribe to new open high findings based on severity.

**unsubscribe_new_open_high_findings_severity**(*client_id=None*)

Unsubscribes the user from new open high findings based on severity

**Parameters**

**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**

dict

**Returns**

The response to the subscription that was performed

**Example**

An example to use unsubscribe_new_open_high_findings_severity()

```
>>> self.[risksenseobject].application_findings.unsubscribe_new_open_high_
↪findings_severity()
```

This helps the user unsubscribe from new open high findings based on severity.

**update_due_date**(*search_filters*, *due_date*, *csvdump=False*, *client_id=None*)

Update the due date for remediation of an application finding.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **due_date** (str) – The due date to assign. Must be in "YYYY-MM-DD" format.
- **csvdump** (bool) – Dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**

int

**Returns**

The job ID in the platform is returned.

**Example**

Lets update an application finding id 1234 to due date 2022-08-11

```
>>> self.[risksenseobject].application_findings.update_due_date([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],'2022-08-11')
```

**Note:** You can also dump the application findings data after updating their due date using `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.update_due_date([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],'2022-08-11',csvdump=True)
```

**self_assign**(*filterfields*, *userid*, *csvdump=False*, *client_id=None*)

The application findings fetched are assigned to the current user

**Parameters**

- **filterfields** (`list`) – A list of dictionaries containing filter parameters.
- **userid** (`list`) – A list of user IDs to be assigned to application findings(s).
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (`bool`) – dumps the data in csv

**Return type**
`int`

**Returns**
The job ID in the platform is returned.

**Example**

Lets assign user *123* to application finding *1234*

```
>>> self.{risksenseobject}.application_finding.self_assign([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],[123])
```

**Note:** You can also dump the application findings data before assigning them to users using `csvdump=True` argument

```
>>> self.{risksenseobject}.application_finding.self_assign([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],[123],csvdump=True)
```

**self_unassign**(*filterfields*, *userid*, *csvdump=False*, *client_id=None*)

The application findings fetched are unassigned from users

**Parameters**

- **filterfields** (`list`) – A list of dictionaries containing filter parameters.
- **userid** (`list`) – A list of user IDs to be assigned to Application Findings(s).
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID in the platform is returned.

---

**Example**

To unassign user 123 from finding id 1234

```
>>> self.{risksenseobject}.application_findings.self_unassign([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],[123])
```

---

**Note:** You can also dump the application findings data before unassigning them from users using `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.self_unassign([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],[123],csvdump=True)
```

---

**add_note**(*search_filters*, *note*, *csvdump=False*, *client_id=None*)

Add a note to application finding(s).

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **note** (`str`) – A note to assign to the application findings.
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID in the platform is returned.

---

**Example**

To add a note 'testing' to application finding id 123

```
>>> self.{risksenseobject}.application_findings.add_note([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"123"}],'testing')
```

---

**Note:** You can also dump the application findings data post adding a note using `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.add_note([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"123"}],'testing',csvdump=True)
```

---

**add_ticket_tag**(*search_filters*, *tag_id*, *client_id=None*)

Adds a ticket tag to the application findings based on a search filter

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – The tag id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform is returned.

---

**Example**

To add a ticket tag to application findings

```
>>> self.{risksenseobject}.application_findings.add_ticket_tag([],123)
```

---

**delete_manage_observations**(*applicationfindingid*, *vulnrequestid*, *csvdump=False*, *client_id=None*)

Delete manage observations

**Parameters**

- **applicationfindingid** (int) – Application finding id
- **vulnrequestid** (int) – Vulnerability request id
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client id of user, if none gets default client id

**Return type**
bool

**Returns**
Success response

---

**Example**

To delete observation linked to application finding id 123 and vulnrequest id 1234

```
>>> self.{risksenseobject}.application_findings.delete_manage_observations(123
↪1234)
```

---

**Note:** You can also dump the application findings data before deleting the manage observation using csvdump=True argument

```
>>> self.{risksenseobject}.application_findings.delete_manage_observations(123
↪1234,csvdump=True)
```

---

**delete**(*search_filters*, *csvdump=False*, *client_id=None*)

Delete application findings based on filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To delete application finding by id 12345

```
>>> self.[risksenseobject].application_findings.delete([{"field":"id","exclusive
↪":False,"operator":"IN","value":"12345"}])
```

---

**Note:** You can also dump the application findings data before deleting the application findings using csvdump=True argument

```
>>> self.[risksenseobject].application_findings.delete([{"field":"id","exclusive
↪":False,"operator":"IN","value":"12345"}],csvdump=True)
```

---

**_tag**(*search_filters*, *tag_id*, *is_remove=False*, *client_id=None*)

Add/Remove a tag to application findings.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – The tag ID to apply.
- **is_remove** (bool) – remove tag? Mention true if need to be removed or false if to add
- **client_id** (typing.Optional[int]) – Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

To add a tag

```
>>> self.[risksenseobject].application_findings._tag([{"field":"id","exclusive
↪":False,"operator":"IN","value":"12345"}],123,is_remove=False)
```

To delete a tag

```
>>> self.rs.application_findings._tag([{"field":"id","exclusive":False,"operator
↪":"IN","value":"12345"}],123,is_remove=True)
```

---

**map_findings**(*filter_request*, *workflowtype*, *workflowuuid*, *csvdump=False*, *client_id=None*)

Maps findings to a worklow request based on workflow uuid.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowtype** (str) – Type of workflow, either falsePositive,remediation,acceptance,severityChange. Please use the exact names as above for workflow type.
- **workflowuuid** (str) – Uuid of the workflow.
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Whether success or not.

---

**Example**

To map a workflow 'st1234' to finding by id '123' of type severitychange

```
>>> self.{risksenseobject}.application_findings.map_findings([{"field":"id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"123"}],'severityChange','st1234')
```

---

**Note:** You can also dump the application findings data post mapping the findings using `csvdump=True` argument

```
>>> self.{risksenseobject}.application_findings.map_findings([{"field":"id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"123"}],'severityChange','st1234',csvdump=True)
```

---

**unmap_findings**(*filter_request*, *workflowtype*, *workflowuuid*, *csvdump=False*, *client_id=None*)

Unmaps findings from worklow request based on workflow uuid.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowtype** (str) – Type of workflow, either falsePositive,remediation,acceptance,severityChange. Please use the exact names as above for workflow type.

- **workflowuuid** (str) – Uuid of the workflow.
- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if its not needed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Whether success or not.

---

**Example**

To unmap a workflow 'st1234' from finding by id '123' of type severitychange

```
>>> self.{risksenseobject}.application_findings.unmap_findings([{"field":"id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"123"}],'severityChange','st1234')
```

---

**Note:** You can also dump the application findings data before unmapping the findings using csvdump=True argument

```
>>> self.{risksenseobject}.application_findings.unmap_findings([{"field":"id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"123"}],'severityChange','st1234',csvdump=True)
```

---

### 1.2.3 Application Url (risksense_api.__subject.__application_urls.__application_urls)

**Application Url module defined for different application url related api endpoints.**

**class** risksense_api.__subject.__application_urls.__application_urls.**ApplicationUrls**(*profile*)

Bases: Subject

ApplicationUrl class

**__init__**(*profile*)

Initialization of ApplicationUrl object.

**Parameters**
**profile** – Profile Object

**get_model**(*client_id=None*)

Get available projections and models for ApplicationUrl.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
ApplicationUrl projections and models are returned.

---

**Examples**

To get model for application url using *get_model()* function

```
>>> self.{risksenseobject}.application_url.get_model()
```

---

**list_applicationurl_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**

- **filter_subject** – Supported Subjects are:
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The JSON output from the platform is returned, listing the available filters.

---

**Examples**

```
>>>  self.{risksenseobject}.application_url.list_applicationurl_filter_fields()
```

---

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
Value suggestions

---

**Example**

To use suggest function is

```
>>> self.{risksenseobject}.application_url.suggest([],{})
```

---

**get_single_search_page**(*search_filters*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns application url data based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – List of dictionaries containing filter parameters.
- **page_num** (int) – Page number of results to be returned.
- **page_size** (int) – Number of results to be returned per page.
- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The paginated JSON response from the platform is returned.

---

**Example**

An example to get single search page of application url data

```
>>> self.{risksenseobject}.application_url.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.{risksenseobject}.application_url.get_single_search_page([],page_num=2
↪page_size=10)
```

---

**search**(*search_filters*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns application url based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **csvdump** (bool) – Dumps the data in csv
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all application url returned by the search using the filter provided.

**Example**

An example to search for application url data is

```
>>> self.{risksenseobject}.application_url.search([])
```

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.{risksenseobject}.application_url.search([],csvdump=True)
```

### 1.2.4 Attachments (risksense_api.__subject.__attachments.__attachments)

**Attachments module defined for different attachments related api endpoints.**

**class** risksense_api.__subject.__attachments.__attachments.**Attachments**(*profile*)

Bases: `Subject`

**Class for Attachments function defintions**.

To utlise Attachments function:

**Parameters**
**profile** (`object`) – Profile Object

**Usage:**
`self.{risksenseobjectname}.attachments.{function}`

**Examples**

To list attachments attached to a tag using *list_attachments()* function

```
>>> self.{risksenseobject}.attachments.list_attachments({tag_id})
```

**__init__**(*profile*)

Initialization of Attachments object.

**Parameters**
**profile** (`object`) – Profile Object

**upload**(*tag_id*, *file_name*, *client_id=None*)

Upload a new attachment for a tag.

**Parameters**

- **tag_id** (`int`) – The tag ID.
- **file_name** (`str`) – The file to be uploaded.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`str`

**Returns**
The UUID of the uploaded file is returned.

---

**Example**

To upload an attachment to a tag 123

```
>>> self.{risksenseobject}.attachments.upload(123 'test.csv')
```

---

`list_attachments`(*tag_id*, *client_id=None*)

List the attachment(s) associated with a tag.

**Parameters**

- **tag_id** (int) – The tag ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The attachments information from the platform is returned.

---

**Example**

To list attachments for tag id 123

```
>>> self.rs.attachments.list_attachments(123)
```

---

`get_attachment`(*tag_id*, *attachment_uuid*, *file_destination*, *client_id=None*)

Get an attachment associated with a tag.

**Parameters**

- **tag_id** (int) – Integer. The tag ID.
- **attachment_uuid** (str) – String. The UUID for the attachment to be downloaded.
- **file_destination** (str) – String. The location to save the attachment locally.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
A True/False is returned reflecting whether or not the operation was successful.

---

**Example**

```
>>> self.rs.attachments.get_attachment(123 "attachmentuuid" "path.csv")
```

---

**delete**(*tag_id*, *attachment_uuid*, *client_id=None*)

Delete an attachment associated with a tag.

**Parameters**

- **tag_id** (int) – The tag ID.
- **attachment_uuid** (str) – The UUID for the attachment to be deleted.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned

---

**Example**

To delete attachment in a tag

```
>>> self.{risksenseobject}.attachments.delete(123 'attachmentuuid123')
```

---

**get_metadata**(*tag_id*, *attachment_uuid*, *client_id=None*)

Get the metadata associated with an attachment.

**Parameters**

- **tag_id** (int) – The tag ID.
- **attachment_uuid** (str) – The UUID for the attachment to be deleted.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform containing the metadata is returned.

---

**Example**

Get attachment metadata

```
>>> self.{risksenseobject}.attachment.get_metada(123 "attachmentuuid123")
```

---

### 1.2.5 Clients (risksense_api.__subject.__clients.__clients)

****Clients module defined for different clients related api endpoints.****

**class** risksense_api.__subject.__clients.__clients.**Clients**(*profile*)

Bases: Subject

**Class for Clients function defintions**.

To utlise clients function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.clients.{function}

---

**Examples**

To get subjects using get_subject() function

```
>>> self.{risksenseobject}.clients.get_subjects()
```

---

**__init__**(*profile*)

Initialization of Clients object.

**Parameters**
**profile** (object) – Profile Object

**get_clients**(*page_size=500*, *page_number=0*)

Gets all clients associated with the API key.

**Parameters**

- **page_size** (int) – Number of results to be returned on each page.
- **page_number** (int) – The page number to be returned.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Example**

To use get_clients method

```
>>> self.{risksenseobject}.get_clients()
```

---

**get_client_info**(*client_id=None*)

Gets the details for a specific client ID.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Example**

To get client info of 123

```
>>> self.{risksenseobject}.geT_client_info(123)
```

---

**get_subjects**(*client_id=None*)

List out all of the available subjects for a specific client.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
The JSON from the platform is returned.

---

**Example**

To try out get_subjects

```
>>> self.{risksenseobject}.get_subjects(1234)
```

---

### 1.2.6 Finding History (risksense_api.__subject.__findinghistory.__findinghistory)

**Finding history module defined for different history related api endpoints.**

**class** risksense_api.__subject.__findinghistory.__findinghistory.**FindingHistory**(*profile*)

Bases: Subject

**Class for Finding History Definitions**.

To utlise Finding history:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.finding_history.{function}

---

**Examples**

To get findings history for an application finding using *get_applicationfinding_history()* function

```
>>> self.{risksenseobject}.finding_history.get_applicationfinding_history(112)
```

---

`__init__`(*profile*)

**Initialization of Finding history Object** .

**Parameters**
**profile** (object) – Profile Object

`get_hostfinding_history`(*vulnerableids*, *client_id=None*)

Get history of hostfindings

**Parameters**

- **vulnerableids** (list) – The vulnerability ids
- **client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
list

**Returns**
The history data

---

**Example**

To get groupby data

```
>>> self.{risksenseobject}.finding_history.get_hostfinding_history([123 123 ])
```

---

`get_applicationfinding_history`(*vulnerableids*, *client_id=None*)

Get history of hostfindings

**Parameters**

- **vulnerableids** (list) – The vulnerability ids
- **client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
list

**Returns**
The history data

---

**Example**

To get groupby data

```
>>> self.{risksenseobject}.finding_history.get_hostfinding_history([123 123 ])
```

---

### 1.2.7 Export (risksense_api.__subject.__exports.__exports)

**Exports module defined for different export related api endpoints.**

**class** risksense_api.__subject.__exports.__exports.**ExportFileType**

Bases: object

ExportFileType class and params

**CSV = 'CSV'**

**XML = 'XML'**

**XLSX = 'XLSX'**

**JSON = 'JSON'**

**class** risksense_api.__subject.__exports.__exports.**ExportRowNumbers**

Bases: object

ExportRowNumbers class and params

**ROW_5000 = '5000'**

**ROW_10000 = '10000'**

**ROW_25000 = '25000'**

**ROW_50000 = '50000'**

**ROW_100000 = '100000'**

**ROW_ALL = 'All'**

**class** risksense_api.__subject.__exports.__exports.**Exports**(*profile*)

Bases: Subject

**Class for Exports function defintions**.

To utlise exports function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.exports.{function}

---

**Examples**

To download an export using *download_export()* function

```
>>> self.{risksenseobject}.exports.download_export(123,'test.csv')
```

---

**__init__**(*profile*)

Initialization of Exports object.

**Parameters**
**profile** (object) – Profile Object

**check_status**(*export_id*, *client_id=None*)

Checks on the status of an export.

**Parameters**

- **export_id** (int) – The ID of the export to be checked.
- **client_id** (typing.Optional[str]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
str

**Returns**
A string reflecting the status of the export is returned.

---

**Example**

To check status of export id 123

```
>>> self.{risksenseobject}.export.check_status(123)
```

---

**download_export**(*export_id*, *filename*, *client_id=None*)

Download an exported file.

**Parameters**

- **export_id** (int) – The ID of the export.
- **filename** (str) – The filename to save the downloaded file as.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
True/False reflects whether or not the download was successful.

---

**Example**

To download an export file

```
>>> self.{risksenseobject}.export.download_export(123, 'test.csv')
```

---

**delete_files**(*export_id*, *client_id=None*)

Delete export job.

**Parameters**

- **export_id** (int) – The export ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
True/False reflecting whether or not the file deletion was successful.

---

**Example**

To delete an export job

```
>>> self.{risksenseobject}.export.delete_files(123)
```

---

### 1.2.8 Groupby (risksense_api.__subject.__groupBy.__groupBy)

**Groupby module defined for different groupby related api endpoints.**

**class** risksense_api.__subject.__groupBy.__groupBy.**GroupBy**(*profile*)

Bases: Subject

Groupby Class

**Parameters**
**profile** (object) –

**__init__**(*profile*)

**Class for Groupby function defintions**.

To utlise groupby function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.groupby.{function}

---

**Examples**

To get hostfindings using groupby method,use *get_groupby_hostfinding()* function

```
>>> self.{risksenseobject}.groupby.get_groupby_hostfinding(123 'test.csv')
```

---

**get_groupby_hostfinding**(*hostfindingkey*, *metricfields*, *filters*, *sortorderfield*, *client_id=None*)

Get groupby values for host finding

**Parameters**

- **hostfindingkey** (str) – The main key where other metric fields depend on
- **metricfields** (list) – The fields that will be populated
- **filters** (list) – The filters which will populate in groupby
- **sortorderfield** (list) – The order to sort the groupby values, please choose ASC for ascending and DESC for descending
- **client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
dict

**Returns**
The groupby data

---

**Example**

To get groupby data

```
>>> self.{risksenseobject}.groupby.get_groupby_hostfinding(["Host Finding Hosts␣
↪Count","Host Finding Open Count","Host Finding Closed Count","Host Finding␣
↪With Threat Count","Host Finding Threat Count"],"Host Finding Asset Tags",[{
↪"field":"criticality","exclusive":false,"operator":"IN","orWithPrevious
↪":false,"implicitFilters":[],"value":"1,"}],[{"field":"Host Finding Asset Tags
↪","direction":"ASC"}])
```

---

**get_groupby_appfinding**(*appfindingkey*, *metricfields*, *filters*, *sortorderfield*, *client_id=None*)

Get groupby values for app finding

**Parameters**

- **appfindingkey** (str) – The main key where other metric fields depend on
- **metricfields** (list) – The fields that will be populated
- **filters** (list) – The filters which will populate in groupby
- **sortorderfield** (list) – The order to sort the groupby values, please choose ASC for ascending and DESC for descending
- **client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Returns**
The groupby data

---

**Example**

To get groupby data

```
>>> self.{risksenseobject}.groupby.get_groupby_appfinding(["App Finding Apps␣
↪Count","App Finding Open Count","App Finding Closed Count","App Finding VRR␣
↪Critical Count","App Finding VRR High Count","App Finding VRR Medium Count",
↪"App Finding VRR Low Count","App Finding VRR Info Count","App Finding␣
↪Severity Critical Count","App Finding Severity High Count","App Finding␣
↪Severity Medium Count","App Finding Severity Low Count","App Finding Severity␣
↪Info Count","App Finding With Threat Count","App Finding Threat Count","App␣
↪Finding CVE Count"],"App Finding Asset Criticality",[{"field":"web_app_url",
↪"exclusive":false,"operator":"IN","orWithPrevious":false,"implicitFilters":[],
↪"value":"Benchmarx_Arnab,"}],[{"field":"App Finding Asset Criticality",
↪"direction":"DESC"}])
```

---

### 1.2.9 Hosts (risksense_api.__subject.__hosts.__hosts)

**Hosts module defined for different hosts related api endpoints.**

**class** risksense_api.__subject.__hosts.__hosts.**Hosts**(*profile*)

Bases: Subject

**Class for Hosts function defintions**.

To utlise Hosts function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.hosts.{function}

---

**Examples**

To get dynamic columns using *getdynamiccolumns()* function

```
>>> self.rs.hosts.getdynamiccolumns()
```

---

**__init__**(*profile*)

**Initialization of Hosts Object** .

**Parameters**
**profile** (object) – Profile Object

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

**Exports and Downloads a file based on the filters defined** .

**Parameters**

- **filename** (str) – Name of the file to export as
- **filters** (list) – Host search filters based on which the export performs
- **client_id** (typing.Optional[int]) – The client id to get the data from. If not supplied, takes default client id

**IGNORE INTERNAL FUNCTION**

---

**Examples**

```
>>> self.{risksenseobject}.hosts.downloadfilterinexport('hostdata',[])
```

---

**create**(*group_id*, *group_ids*, *assessment_id*, *network_id*, *ip_address*, *hostname*, *subnet*, *disc_date*, *client_id=None*, *scannerFirstDiscoveredOn=None*, *scannerlastDiscoveredOn=None*, *services=None*, *criticality=None*, *os_scanner=None*, *createcmdb=None*, *lockCmdb=None*)

Creates a host based on the data provided by the user.

**Parameters**

- **group_id** (int) – Group ID
- **group_ids** (list) – Group IDs

- **assessment_id** (int) – Assessment ID
- **network_id** (int) – Network ID
- **ip_address** (str) – IP Address of host
- **hostname** (str) – Hostname
- **subnet** (str) – Subnet host belongs to
- **disc_date** (str) – Discovered Date (Date formatted as "YYYY-MM-DDTHH:MM:SS")
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **scannerFirstDiscoveredOn** (typing.Optional[str]) –
- **scannerlastDiscoveredOn** (typing.Optional[str]) –
- **services** (typing.Optional[list]) –
- **criticality** (typing.Optional[int]) –
- **os_scanner** (typing.Optional[int]) –
- **createcmdb** (typing.Optional[dict]) –
- **lockCmdb** (typing.Optional[dict]) –

**Keyword Arguments**

- **scanner_first_discovered_on** (str) – Scanner First Discovered On
- **scanner_last_discovered_on** (str) – Scanner Last Discovered On
- **criticality** (int) – int 1-5
- **services** (list) – list A list of dicts, each dict containing portNumber(int), and name (str)
- **os_scanner** (*str*) – dict A dict containing name (str), family(str), :obj: *class(str)*, vendor(str), product (str), and certainty (int)
- **createcmdb** (*dict*) – dict
- **lockCmdb** (*dict*) – dict

**Return type**
int

**Returns**
The host ID on the platform is returned.

**getdynamiccolumns**(*client_id=None*)

Gets Dynamic columns for the hosts.

**Parameters**
**client_id** (typing.Optional[int]) – If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Dynamic columns

**Examples**

```
>>> self.{risksenseobject}.hosts.getdynamiccolumns()
```

`list_host_filter_fields`(*client_id=None*)

Lists all the host filter data from the platform

**Parameters**
`client_id` (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`list`

**Returns**
The JSON output from the platform is returned, listing the available filters.

**Examples**

```
>>> self.{risksenseobject}.hosts.list_host_filter_fields()
```

`delete`(*search_filters*, *csvdump=False*, *client_id=None*)

Delete hosts based on provided filters.

**Parameters**

- `search_filters` (`list`) – A list of dictionaries containing filter parameters.
- `csvdump` (`bool`) – dumps the data in csv
- `client_id` (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The delete Job ID

**Examples**

To delete a host:

```
>>> self.{risksenseobject}.hosts.delete([])
```

**Note:** You can also dump the data of the hosts that are going to be deleted in a csv file using `csvdump=True` argument:

```
>>> self.{risksenseobject}.hosts.delete([],csvdump=True)
```

`get_groupby_host`(*client_id=None*)

Gets all the groupby fields for hosts

**Parameters**
**client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
dict

**Returns**
The group by key metrics

**IGNORE INTERNAL FUNCTION** .. admonition:: Example

```
>>> self.{risksenseobject}.hosts.get_groupby_host()
```

---

**Note:** This function just returns the groupbyfields

---

**post_groupby_host**(*filters=[]*, *sortorder=None*, *csvdump=False*, *client_id=None*)

Gets the groupby values for hosts based on the filter provided

**Parameters**

- **filters** (list) – The filters which will populate in groupby
- **sortorder** (typing.Optional[str]) – The order to sort the groupby values, please choose ASC for ascending and DESC for descending
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – The client id , if none, default client id is taken

**Return type**
dict

**Returns**
The hosts data grouped based on the particular field provided

---

**Example**

```
>>> self.{risksenseobject}.hosts.post_groupby_host({filter})
```

The filter must be provided for the group by to be used. The groupby fields will be displayed in the *terminal* and you must choose a *group by* filter to which the data will be populated

---

**Note:** This function also has an option to dump the data in a csv by a simple argument, csvdump=True

```
>>> self.{risksenseobject}.hosts.post_groupby_host({filter},csvdump=True)
```

---

**update_hosts_attrs**(*search_filters*, *csvdump=False*, *client_id=None*, ***kwargs*)

This function updates hosts attributes based on search filters

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – dumps the data in csv

**Keyword Arguments**

- **ip_address** (str) – IP Address of host
- **hostname** (str) – Hostname
- **subnet** (str) – Subnet host belongs to
- **discovered_date** (str) – Date formatted as "YYYY-MM-DD"
- **criticality** (int) – 1-5
- **services** (int) – A list of dicts, each dict containing portNumber(int), and name (str)
- **os_scanner** (dict) – A dict containing name(str), family (str), class(str), vendor(str), product(str), and certainty(int)

**Return type**
int

**Returns**
The host ID on the platform is returned.

---

**Example**

```
>>> self.{risksenseobject}.hosts.update_hosts_attrs([],criticality=2)
```

An example to change the host attributes based on ip address

---

**Note:** You can also dump the job id data in a csv by simply using csvdump=True argument

```
>>> self.{risksenseobject}.hosts.update_hosts_attrs([],criticality=3
↪csvdump=True)
```

---

**update_hosts_cmdb**(*search_filters*, *client_id=None*, ***kwargs*)

Updates host cmdb

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Keyword Arguments**

- **manufacturer** (str) – Manufacturer
- **model_id** (str) – Model id
- **mac_address** (str) – Mac Address
- **location** (str) – Location
- **managed_by** (str) – Managed By
- **owned_by** (str) – Owned By
- **supported_by** (str) – Supported By
- **support_group** (str) – Support Group

- **sys_id** (str) – Sys id
- **os** (str) – Operating System
- **last_scan_date** (str) – Date formatted as "YYYY-MM-DD"
- **asset_tag** (str) – Asset Tag
- **ferpa** (bool) – Ferpa
- **hipaa** (bool) – Hipaa
- **pci** (bool) – PCI
- **cf_1** (str) – Custom field_1
- **cf_2** (str) – Custom field_2
- **cf_3** (str) – Custom field_3
- **cf_4** (str) – Custom field_4
- **cf_5** (str) – Custom field_5
- **cf_6** (str) – Custom field_6
- **cf_7** (str) – Custom field_7
- **cf_8** (str) – Custom field_8
- **cf_9** (str) – Custom field_9
- **cf_10** (str) – Custom field_10
- **am_1** (str) – Asset Matching field_1
- **am_2** (str) – Asset Matching field_2
- **am_3** (str) – Asset Matching field_3

**Return type**
int

**Returns**
The job ID

---

**Example**

An example to update hosts cmdb with manufacturer name or model id

```
>>> self.{risksenseobject}.hosts.update_hosts_cmdb([],manufacturer=
↪'manufacturername',model_id='R1234')
```

Use the keyword arguments depending on what cmdb data you need to update

---

**lock_hosts_cmdb**(*search_filters*, *client_id=None*, ***kwargs*)

Locks The hosts cmdb data

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Keyword Arguments**

- **manufacturer** (str) – Manufacturer
- **business_criticality** (int) – business criticality
- **model_id** (str) – Model id
- **mac_address** (str) – Mac Address
- **location** (str) – Location
- **managed_by** (str) – Managed By
- **owned_by** (str) – Owned By
- **supported_by** (str) – Supported By
- **support_group** (str) – Support Group
- **sys_id** (str) – Sys id
- **os** (str) – Operating System
- **last_scan_date** (str) – Date formatted as "YYYY-MM-DD"
- **asset_tag** (str) – Asset Tag
- **ferpa** (bool) – Ferpa
- **hipaa** (bool) – Hipaa
- **pci** (bool) – PCI
- **cf_1** (str) – Custom field_1
- **cf_2** (str) – Custom field_2
- **cf_3** (str) – Custom field_3
- **cf_4** (str) – Custom field_4
- **cf_5** (str) – Custom field_5
- **cf_6** (str) – Custom field_6
- **cf_7** (str) – Custom field_7
- **cf_8** (str) – Custom field_8
- **cf_9** (str) – Custom field_9
- **cf_10** (str) – Custom field_10
- **am_1** (str) – Asset Matching field_1
- **am_2** (str) – Asset Matching field_2
- **am_3** (str) – Asset Matching field_3

**Returns**
The job ID

---

**Example**

An example to lock hosts cmdb with manufacturer name or model id

```
>>> self.{risksenseobject}.hosts.lock_hosts_cmdb([],business_criticality=437)
```

Use the keyword arguments depending on what cmdb data you need to lock

---

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns hosts based on the provided filter(s) and other parameters. This gets paginated results data

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_num** (int) – The page number of results to be returned.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Example**

An example to get single search page of hosts data

```
>>> self.{risksenseobject}.hosts.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.{risksenseobject}.hosts.get_single_search_page([],page_num=2,page_
↪size=10)
```

---

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns hosts based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC

- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all hosts returned by the search using the filter provided.

---

**Example**

An example to search for host data is

```
>>> self.{risksenseobject}.hosts.search([])
```

Where [] is the search filter for all hosts, you can provide your search filter there.

---

**Note:** You can also dump the search based data in a csv by simply providing csvdump=True argument

```
>>> self.{risksenseobject}.hosts.search([],csvdump=True)
```

---

get_count(*search_filters*, *client_id=None*)

Gets a count of hosts identified using the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The number of hosts identified using the provided filter(s).

---

**Example**

An example to use get count function is as follows

```
>>> self.{risksenseobject}.hosts.get_count([])
```

Where [] is the search filter for all hosts, you can provide your search filter there.

---

add_tag(*search_filters*, *tag_id*, *csvdump=False*, *client_id=None*)

Adds a tag to host(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – ID of tag to tbe added to host(s).
- **csvdump** (bool) – dumps the data in csv

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

An example to add a tag is

```
>>> self.{risksenseobject}.hosts.add_tag([],1234)
```

Where

[] is the search filter for all hosts, you can provide your search filter there.

1234 is the tag id

---

**Note:** You can also dump the hosts from the search filters post the tag completion for more information by csvdump=True argument

```
>>> self.{risksenseobject}.hosts.add_tag([],1234,csvdump=True)
```

---

**remove_tag**(*search_filters*, *tag_id*, *csvdump=False*, *client_id=None*)

Removes a tag from host(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – ID of tag to be removed from host(s).
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – dumps the data in csv

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

An example to use remove tag is

```
>>> self.{risksenseobject}.hosts.remove_tag([],123)
```

Where

[] is the search filter for all hosts, you can provide your search filter there.

123 is the tag id

---

**Note:** You can also dump the hosts which the tags will be removed from with a `csvdump=True` argument

```
>>> self.[risksenseobject].hosts.remove_tag([],123,csvdump=True)
```

`getexporttemplate`(*client_id=None*)

Gets configurable export template for Hosts.

**Parameters**
**client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`list`

**Returns**
The Exportable fields

**Example**

An example to use getexporttemplate

```
>>> self.[risksenseobject].hosts.getexporttemplate()
```

This gets all the export templates for hosts

`merge_host`(*search_filter*, *host_id_to_merge_to*, *csvdump=False*, *client_id=None*)

Merges host(s).

**Parameters**

- **search_filter** (`list`) – A list of dictionaries containing filter parameters.
- **host_id_to_merge_to** (`int`) – The host id to which the hosts based on the filter will be merged to
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, it will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

**Example**

An example to use merge_host is

```
>>> self.[risksenseobject].hosts.merge_host([],123)
```

Where

`[]` is the search filter for all hosts, you can provide your search filter there.

`123` is the host id to which the hosts will be merged to.

**Note:** You can also dump the hosts that are going to be merged before merging them by `csvdump=True` argument

```
>>> self.{risksenseobject}.hosts.merge_host([],123,csvdump=True)
```

`set_asset_criticality`(*filter*, *assetcriticality*, *csvdump=False*, *client_id=None*)

Sets asset criticality of the host.

**Parameters**

- **filter** (`list`) – Search filters
- **assetcriticality** (`int`) – The asset criticality to provide.
- **csvdump** (`bool`) – Dump the csv data.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

**Example**

An example to use set_asset_criticality is

```
>>> self.{risksenseobject}.hosts.set_asset_criticality([],4)
```

Where

`[]` is the search filter for all hosts, you can provide your search filter there.

4 is the criticality of the asset to set to

**Note:** You can also dump the hosts to which asset criticality should be changed by `csvdump=True` argument

```
>>> self.{risksenseobject}.hosts.set_asset_criticality([],4,csvdump=True)
```

`set_address_type`(*filter*, *addresstype*, *csvdump=False*, *client_id=None*)

Sets address type of the host.

**Parameters**

- **filter** (`list`) – Search filters
- **addresstype** (`str`) – Provide external for external address and internal for internal
- **csvdump** (`bool`) – Dump the csv data.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

An example to use set_address_type is

```
>>> self.{risksenseobject}.hosts.set_address_type([],'external')
```

Where

`[]` is the search filter for all hosts, you can provide your search filter there.

`external` is to set the address type as external address.

---

**Note:** You can also dump the hosts which the address type will be set by `csvdump=True` argument

```
>>> self.{risksenseobject}.hosts.set_address_type([],'external',csvdump=True)
```

---

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*)

Initiates an export job on the platform for host(s) based on the provided filter(s).

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **file_name** (`str`) – The name to be used for the exported file.
- **row_count** (`str`) – No of rows to be exported. Available options ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **file_type** (`str`) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID in the platform from is returned.

---

**Example**

An example to use export is

```
>>> self.{risksenseobject}.hosts.export([],'testingexport')
```

Where

`[]` is the search filter for all hosts, you can provide your search filter there.

`testingexport` is the filename to export the file to

You can change the filetype to any of the names above or even the other positional arguments as mentioned

```
>>> self.[risksenseobject].hosts.export([],'testingexport',file_
↪type=ExportFileType.JSON)
```

---

**network_move**(*search_filters*, *network_identifier*, *is_force_merge=False*, *csvdump=False*, *client_id=None*)

Moves host(s) into a new network as specified.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **network_identifier** (int) – Network ID to move the hosts to
- **is_force_merge** (bool) – Force merge of hosts?
- **csvdump** (bool) – Dump the csv data.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

An example to use network_move is

```
>>> self.[risksenseobject].hosts.network_move([],12345,False)
```

Where

`[]` is the search filter for all hosts, you can provide your search filter there.

`12345` is the network id to which the hosts will move to

`False` is to not force merge the hosts

---

**Note:** You can also dump the hosts that are going to be moved before moving them by `csvdump=True` argument

```
>>> self.[risksenseobject].hosts.network_move([],12345,False,csvdump=True)
```

---

**run_urba**(*search_filters*, *csvdump=False*, *client_id=None*)

Initiates the update of remediation by assessment for hosts specified in filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **csvdump** (bool) – Dump the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, it will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

An example to use run_urba is

```
>>> self.{risksenseobject}.hosts.run_urba([])
```

Where

`[]` is the search filter for all hosts, you can provide your search filter there.

This will run the urba for all the hosts fetched from the search filter

---

**Note:** You can also dump the hosts to which urba is being run by `csvdump=True` argument

```
>>> self.{risksenseobject}.hosts.run_urba([],csvdump=True)
```

---

**add_note**(*search_filters*, *new_note*, *csvdump=False*, *client_id=None*)

Adds a note to host(s) based on the filter(s) provided.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **new_note** (`str`) – The note to be added to the host(s).
- **csvdump** (`bool`) – Dump the csv data.
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

An example to use add_note is

```
>>> self.{risksenseobject}.hosts.add_note([],'test')
```

Where

`[]` is the search filter for all hosts, you can provide your search filter there.

`test` is the note which will be given to the hosts

---

**Note:** You can also dump the hosts to which notes will be added post adding the note by `csvdump=True` argument

```
>>> self.{risksenseobject}.hosts.add_note([],'test',csvdump=True)
```

**get_model**(*client_id=None*)

Get available projections and models for Hosts.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Hosts projections and models are returned.

**Example**

An example to use get_model is

```
>>> self.{risksenseobject}.hosts.get_model()
```

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
Value suggestions

**Example**

To use suggest function is

```
>>> self.{risksenseobject}.hosts.suggest([],{})
```

Where

[] is the first search filter

{} is the seconf search filter

**add_group**(*search_filter*, *group_ids*, *csvdump=False*, *client_id=None*)

Add host(s) to one or more groups.

**Parameters**

- **search_filter** (list) – Search filter
- **group_ids** (list) – List of Group IDs to add to host(s).

- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
int

**Returns**
Job ID of group add job

---

**Example**

An example to use add_group is

```
>>> self.[risksenseobject].hosts.add_group([],[2,3,4])
```

Where

[] is the search filter for all hosts, you can provide your search filter there.

[2,3,4] are the group ids to add the hosts to .

---

**Note:** You can also dump the hosts which will be addedd to the groups by csvdump=True argument

```
>>> self.[risksenseobject].hosts.add_group([],[2,3,4],csvdump=True)
```

---

**remove_group**(*search_filter*, *group_ids*, *csvdump=False*, *client_id=None*)

Remove host(s) from one or more groups.

**Parameters**

- **search_filter** (list) – Search filter
- **group_ids** (list) – List of Group IDs to add to host(s).
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
int

**Returns**
Job ID of group remove job

---

**Example**

An example to use remove_group is

```
>>> self.[risksenseobject].hosts.remove_group([],[2,3,4])
```

Where

[] is the search filter for all hosts, you can provide your search filter there.

[2,3,4] are the group ids to remove the hosts from .

**Note:** You can also dump the hosts which will be removed from the groups by `csvdump=True` argument

```
>>> self.{risksenseobject}.hosts.remove_group([],[2,3,4],csvdump=True)
```

risksense_api.__subject.__hosts.__hosts.**apply_system_filters**(*self*, *csvdump=False*, *client_id=None*)

Get data of the hosts based on system filter.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – dumps the data in csv

**Return type**
list

**Returns**
The data of the system filter based host values are returned

**Example**

An example to use apply_system_filters is

```
>>> self.{risksenseobject}.hosts.apply_system_filters()
```

Where

[] is the search filter for all hosts, you can provide your search filter there.

The system filters will be displayed in the terminal to which you must provide a key value and the data returned will reflect based on the system filter chosrn

**Note:** You can also dump the hosts from the system filters search by `csvdump=True` argument

```
>>> self.{risksenseobject}.hosts.apply_system_filters(csvdump=True)
```

### 1.2.10 Host Findings (risksense_api.__subject.__host_findings.__host_findings)

**Host Findings module defined for different host findings related api endpoints.**

class risksense_api.__subject.__host_findings.__host_findings.**HostFindings**(*profile*)

Bases: Subject

**Class for HostFindings function defintions**.

To utlise Host Findings function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**

```
self.{risksenseobjectname}.host_findings.{function}
```

---

**Examples**

To get model for host findings using *get_model()* function

```
>>> self.{risksenseobject}.host_findings.get_model()
```

---

**__init__**(*profile*)

Initialization of HostFindings object.

profile: Profile Object :type profile: _profile

**Parameters**

**profile** (object) –

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

**Exports and Downloads a file based on the filters defined** .

**Parameters**

- **filename** (str) – Name of the file to export as
- **filters** (list) – host findings search filters based on which the export performs
- **client_id** (typing.Optional[int]) – The client id to get the data from. If not supplied, takes default client id

**IGNORE INTERNAL FUNCTION**

---

**Examples**

```
>>> self.{risksenseobject}.host_findings.downloadfilterinexport(
↪'hostfindingsdata',[])
```

---

**create**(*host_id_list*, *assessment_id*, *severity*, *source_id*, *scanner_uuid*, *title*, *finding_type*, *synopsis*, *description*, *solution*, *service_name*, *service_portnumber*, *cveids=[]*, *filters=[]*, *csvdump=False*, *client_id=None*)

Manually create a new host finding.

**Parameters**

- **host_id_list** (list) – List of Host IDs to associate with this finding
- **assessment_id** (int) – Assessment ID
- **severity** (str) – Severity
- **source_id** (str) – Source ID
- **scanner_uuid** (str) – Scanner UUID
- **title** (str) – Host Finding Title
- **finding_type** (str) – Host Finding Type
- **synopsis** (str) – Synopsis
- **description** (str) – Description

- **solution** (str) – Solution
- **service_name** (str) – Service name
- **cveids** (list) – Ids of cves
- **service_portnumber** (str) – Service portnumber
- **filters** (list) – A series of filters that make up a complete filter
- **csvdump** (bool) – dumps id to csv
- **client_id** – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

Creating host finding

```
>>> self.{risksenseobject}.host_findings.create([123],190805,"9","publicnew",
↪"85200f98-1ea6-4641-9d27-171dc79f693f","something",'SERVICE',"testing to work␣
↪on",'somethingto work on','something to work on','new','5',[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"6371904"}])
```

---

**Note:** You can also dump the host finding job id created in a csv using `csvdump=True`:

```
>>> self.{risksenseobject}.host_findings.create([123],190805,"9","publicnew",
↪"85200f98-1ea6-4641-9d27-171dc79f693f","something",'SERVICE',"testing to work␣
↪on",'somethingto work on','something to work on','new','5',[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"6371904"}],csvdump=True)
```

---

**update**(*hostfindingid*, *client_id=None*, *csvdump=False*, ***kwargs*)

Update a new host finding.

**Parameters**

- **hostfindingid** (int) – Host finding id which you want to update
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – dumps id to csv

**Keyword Arguments**

- **title** (str) – title
- **description)** (str) – description
- **synopsis** (str) – synopsis
- **solution** (str) – solution

**Return type**
int

**Returns**
The hostfinding ID is returned.

---

**Example**

To update host finding id 123's description to 'new description'

```
>>> self.rs.host_findings.update(123,description='new description')
```

---

**Note:** You can also dump the host finding job id updated in a csv using `csvdump=True`:

```
>>> self.rs.host_findings.update(123,description='new description',csvdump=True)
```

---

`delete_manage_observations`(*hostfindingid*, *asssessmentid*, *csvdump=False*, *client_id=None*)

Delete manage observations

**Parameters**

- **hostfindingid** (`int`) – Host finding id
- **asssessmentid** (`list`) – Assessment id
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client id of user, if none gets default client id

**Returns**
The jsonified response

**Return type**
jsonified_response

---

**Example**

To delete observation linked to host finding id 123 and assessment id 1234

```
>>> self.[risksenseobject].host_findings.delete_manage_observations(123,[1234])
```

---

**Note:** You can also dump the host findings data before deleting the manage observation using `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.delete_manage_observations(123,[1234],
↪csvdump=True)
```

---

`get_hostfinding_history`(*vulnerableids*, *csvdump=True*, *client_id=None*)

Get Host findings history

**Parameters**

- **vulnerableids** (`list`) – List of vulnerable ids to get history of
- **client_id** (`typing.Optional[int]`) – The client id , if none, default client id is taken
- **csvdump** – dumps the data in csv

**Return type**
`list`

**Returns**
The jsonified response

---

**Example**

To get host finding history

---

`get_groupby_hostfinding`(*client_id=None*)

Gets all the groupby key metrics for host findings

**Parameters**
**`client_id`** (`typing.Optional[int]`) – The client id , if none, default client id is taken

**Return type**
`dict`

**Returns**
The group by key metrics

---

**Example**

```
>>> self.{risksenseobject}.host_findings.get_groupby_hostfinding()
```

---

**IGNORE INTERNAL FUNCTION** .. note:: This function just returns the groupbyfields

`groupby_hostfinding`(*filters=[]*, *sortorder=None*, *client_id=None*, *csvdump=False*)

Get groupby values for host finding

**Parameters**

- **`filters`** (`list`) – The filters which will populate in groupby
- **`sortorder`** (`typing.Optional[str]`) – The order to sort the groupby values, please choose ASC for ascending and DESC for descending
- **`client_id`** (`typing.Optional[int]`) – The client id , if none, default client id is taken
- **`csvdump`** (`bool`) – dumps the data in csv

**Returns**
Group by information

**Return type**
groupby

---

**Example**

```
>>> self.{risksenseobject}.host_findings.groupby_hostfinding({filter})
```

The filter must be provided for the group by to be used. The groupby fields will be displayed in the *terminal* and you must choose a *group by* filter to which the data will be populated

---

**Note:** This function also has an option to dump the data in a csv by a simple argument, `csvdump=True`

```
>>> self.{risksenseobject}.host_findings.groupby_hostfinding({filter},
↪csvdump=True)
```

---

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*, *csvdump=False*)

Searches for and returns hostfindings based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_num** (int) – The page number of results to be returned.
- **csvdump** (bool) – Dumps the data in csv
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Example**

An example to get single search page of host findings data

```
>>> self.{risksenseobject}.host_findings.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.{risksenseobject}.host_findings.get_single_search_page([],page_num=2
↪page_size=10)
```

---

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns hostfindings based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL

- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **csvdump** (bool) – dumps data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all host findings returned by the search using the filter provided.

---

**Example**

An example to search for host finding data is

```
>>> self.[risksenseobject].host_findings.search([])
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.search([],csvdump=True)
```

---

**get_count**(*search_filters*, *client_id=None*)

Gets a count of hostfindings identified using the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The number of hostfindings identified using the provided filter(s).

---

**Example**

An example to use get count function is as follows

```
>>> self.[risksenseobject].host_findings.get_count([])
```

---

**add_tag**(*search_filters*, *tag_id*, *csvdump=False*, *client_id=None*)

Adds a tag to hostfinding(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.

- **tag_id** (int) – ID of tag to tbe added to hostfinding(s).
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

An example to add a tag is

```
>>> self.[risksenseobject].host_findings.add_tag([],1234)
```

---

**Note:** You can also dump the host findings from the search filters post the tag completion for more information by `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.add_tag([],1234,csvdump=True)
```

---

**remove_tag**(*search_filters*, *tag_id*, *client_id=None*, *csvdump=False*)

Removes a tag from hostfinding(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – ID of tag to tbe removed from hostfinding(s).
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – dumps the data in csv

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

An example to use remove tag is

```
>>> self.[risksenseobject].host_findings.remove_tag([],123)
```

---

**Note:** You can also dump the host findings which the tags will be removed from with a `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.remove_tag([],123,csvdump=True)
```

---

**assign**(*search_filters*, *users*, *csvdump=False*, *client_id=None*)

Assigns hostfinding(s) to a list of user IDs.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **users** (list) – A list of user IDs to be assigned to hostfinding(s).
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

Lets assign user *123* to host findings based on filter of patch id *123*

```
>>> self.[risksenseobject].host_findings.assign([{"field":"source_patch_ids",
↪"exclusive":False,"operator":"IN","value":"123"}],[123])
```

---

**Note:** You can also dump the host findings data before assigning them to users using `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.assign([{"field":"source_patch_ids",
↪"exclusive":False,"operator":"IN","value":"123"}],[123],csvdump=True)
```

---

**unassign**(*search_filters*, *users*, *csvdump=False*, *client_id=None*)

Unassigns hostfinding(s) from a list of user IDs.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **users** (list) – A list of user IDs to be unassigned from hostfinding(s).
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

Lets unassign user *123* from host findings based on filter of patch id *123*

```
>>> self.[risksenseobject].host_findings.unassign([{"field":"source_patch_ids",
↪"exclusive":False,"operator":"IN","value":"123"}],[123])
```

---

**Note:** You can also dump the host findings data before unassigning them from users using `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.unassign([{"field":"source_patch_ids",
↪"exclusive":False,"operator":"IN","value":"123"}],[123],csvdump=True)
```

---

**self_assign**(*filterfields*, *userid*, *csvdump=False*, *client_id=None*)

The host findings fetched are assigned to the current user

**Parameters**

- **filterfields** (`list`) – A list of dictionaries containing filter parameters.
- **csvdump** (`bool`) – dumps the data in csv
- **userid** (`list`) – A list of user IDs to be assigned to hostfinding(s).
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID in the platform is returned.

---

**Example**

Lets assign user *123* to host finding *1234*

```
>>> self.[risksenseobject].host_findings.self_assign([{"field":"id","exclusive
↪":False,"operator":"IN","value":"1234"}],[123])
```

---

**Note:** You can also dump the host findings data before assigning them to users using `csvdump=True` argument

```
>>> self.[risksenseobject].host_finding.self_assign([{"field":"id","exclusive
↪":False,"operator":"IN","value":"1234"}],[123],csvdump=True)
```

---

**self_unassign**(*filterfields*, *userids*, *client_id=None*, *csvdump=False*)

The host findings fetched are unassigned from the current user

**Parameters**

- **filterfields** (`list`) – A list of dictionaries containing filter parameters.
- **userids** (`list`) – A list of integers of user ids
- **csvdump** (`bool`) – dumps the data in csv

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform is returned.

---

**Example**

To unassign user 123 from finding id 1234

```
>>> self.[risksenseobject].host_findings.self_unassign([{"field":"id","exclusive
↪":False,"operator":"IN","value":"1234"}],[123])
```

---

**Note:** You can also dump the host findings data before unassigning them from users using `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.self_unassign([{"field":"id","exclusive
↪":False,"operator":"IN","value":"1234"}],[123],csvdump=True)
```

---

**list_hostfinding_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The JSON output from the platform is returned, listing the available filters.

---

**Examples**

```
>>>  self.[risksenseobject].host_findings.list_hostfinding_filter_fields()
```

---

**getexporttemplate**(*client_id=None*)

Gets configurable export template for host findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

---

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.host_findings.getexporttemplate()
```

This gets all the export templates for host findings

---

**getexporttemplatebyid**(*export_id=None*, *client_id=None*)

Gets configurable export template for host findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

---

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.host_findings.getexporttemplate()
```

This gets all the export templates for host findings

---

**Parameters**
**export_id** (typing.Optional[int]) –

**getexporttemplates**(*client_id=None*)

Gets created existing export template for host findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

---

**Example**

An example to use getexporttemplates

```
>>> self.{risksenseobject}.host_findings.getexporttemplates()
```

This gets all the export templates for host findings

---

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *export_id=None*, *client_id=None*)

Initiates an export job on the platform for host finding(s) based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Available options ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **exportable_filter** – Exportable filter
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **export_id** (typing.Optional[int]) – If present, an export template id of the template to use to export.

**Return type**
int

**Returns**
The job ID in the platform from is returned.

---

**Example**

An example to use export is

```
>>> self.{risksenseobject}.host_findings.export([],'testingexport')
```

You can change the filetype to any of the names above or even the other positional arguments as mentioned

```
>>> self.{risksenseobject}.host_findings.export([],'testingexport',file_
↪type=ExportFileType.JSON)
```

---

**update_due_date**(*search_filters*, *new_due_date*, *csvdump=False*, *client_id=None*)

Updates the due date assigned to hostfinding(s) based on the provided filter(s)

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **new_due_date** (str) – The new due date in the "YYYY-MM-DD" format.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

---

**Example**

Lets update an host finding id 1234 to due date 2022-08-11

```
>>> self.[risksenseobject].host_findings.update_due_date([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],'2022-08-11')
```

**Note:** You can also dump the host findings data after updating their due date using `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.update_due_date([{"field":"id",
↪"exclusive":False,"operator":"IN","value":"1234"}],'2022-08-11',csvdump=True)
```

**add_note**(*search_filters*, *new_note*, *csvdump=False*, *client_id=None*)

Adds a note to hostfinding(s) based on the filter(s) provided.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **new_note** (`str`) – The note to be added to the hostfinding(s). String.
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

**Example**

To add a note 'testing' to host finding id 123

```
>>> self.[risksenseobject].host_findings.add_note([{"field":"id","exclusive
↪":False,"operator":"IN","value":"123"}],'testing')
```

**Note:** You can also dump the host findings data post adding a note using `csvdump=True` argument

```
>>> self.[risksenseobject].host_findings.add_note([{"field":"id","exclusive
↪":False,"operator":"IN","value":"123"}],'testing',csvdump=True)
```

**delete**(*search_filters*, *csvdump=False*, *client_id=None*)

Deletes hostfinding(s) based on the provided filter(s)

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Example**

To delete host finding by id 12345

```
>>> self.{risksenseobject}.host_findings.delete([{"field":"id","exclusive
↪" False,"operator":"IN","value":"12345"}])
```

---

**Note:** You can also dump the host findings data before deleting the host findings using `csvdump=True` argument

```
>>> self.{risksenseobject}.host_findings.delete([{"field":"id","exclusive
↪" False,"operator":"IN","value":"12345"}],csvdump=True)
```

---

**subscribe_new_open_ransomware_findings**(*client_id=None*)

Subscribes the user to new open ransomware findings

**Parameters**
**client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`dict`

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use subscribe_new_open_ransomware_findings()

```
>>> self.{risksenseobject}.host_findings.subscribe_new_open_ransomware_
↪findings()
```

This helps the user subscrive to new open ransomware findings

---

**unsubscribe_new_open_ransomware_findings**(*client_id=None*)

Unsubscribes the user from new open ransomware findings

**Parameters**
**client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`dict`

**Returns**
The response to the unsubscription that was performed

**Example**

An example to use unsubscribe_new_open_ransomware_findings()

```
>>> self.[risksenseobject].host_findings.unsubscribe_new_open_ransomware_
↪findings()
```

This helps the user unsubscribe from new open ransomware findings

**subscribe_new_open_critical_findings_vrr**(*client_id=None*)

Subscribes the user to new open critical findings based on vrr

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

**Example**

An example to use subscribe_new_open_critical_findings_vrr()

```
>>> self.[risksenseobject].host_findings.subscribe_new_open_critical_findings_
↪vrr()
```

This helps the user subscribe to new open critical findings based on vrr.

**unsubscribe_new_open_critical_findings_vrr**(*client_id=None*)

Unsubscribes the user from new open critical findings based on vrr

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

**Example**

An example to use unsubscribe_new_open_critical_findings_vrr()

```
>>> self.[risksenseobject].host_findings.unsubscribe_new_open_critical_findings_
↪vrr()
```

This helps the user to unsubscribe from new open critical findings based on vrr.

**subscribe_new_open_critical_findings_severity**(*client_id=None*)

Subscribes the user to new open critical findings based on severity

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use subscribe_new_open_critical_findings_severity()

```
>>> self.{risksenseobject}.host_findings.subscribe_new_open_critical_findings_
↪severity()
```

This helps the user subscribe to new open critical findings based on severity.

---

**unsubscribe_new_open_critical_findings_severity**(*client_id=None*)

Unsubscribes the user from new open critical findings based on severity

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use unsubscribe_new_open_critical_findings_severity()

```
>>> self.{risksenseobject}.host_findings.unsubscribe_new_open_critical_findings_
↪severity()
```

This helps the user unsubscribe from new open critical findings based on severity.

---

**subscribe_new_open_high_findings_vrr**(*client_id=None*)

Subscribes the user to new open high findings based on vrr

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

**Example**

An example to use subscribe_new_open_high_findings_vrr()

```
>>> self.{risksenseobject}.host_findings.subscribe_new_open_high_findings_vrr()
```

This helps the user subscribe to new open high findings based on vrr.

**unsubscribe_new_open_high_findings_vrr**(*client_id=None*)

Unsubscribe the user from new open high findings based on vrr

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

**Example**

An example to use unsubscribe_new_open_high_findings_vrr()

```
>>> self.{risksenseobject}.host_findings.unsubscribe_new_open_high_findings_
↪vrr()
```

This helps the user unsubscribe from new open high findings based on vrr.

**subscribe_new_open_high_findings_severity**(*client_id=None*)

Subscribes the user to new open high findings based on severity

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

**Example**

An example to use subscribe_new_open_high_findings_severity()

```
>>> self.{risksenseobject}.host_findings.subscribe_new_open_high_findings_
↪severity()
```

This helps the user subscribe to new open high findings based on severity.

**unsubscribe_new_open_high_findings_severity**(*client_id=None*)

Unsubscribes the user from new open high findings based on severity

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The response to the subscription that was performed

---

**Example**

An example to use unsubscribe_new_open_high_findings_severity()

```
>>> self.{risksenseobject}.host_findings.unsubscribe_new_open_high_findings_
↪severity()
```

This helps the user unsubscribe from new open high findings based on severity.

---

**map_findings**(*filter_request*, *workflowtype*, *workflowuuid*, *csvdump=False*, *client_id=None*)

Map hostfindings to a workflow .

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowtype** (str) – Type of workflow, either falsePositive,remediation,acceptance,severityChange. Please use the exact names as above for workflow type
- **workflowuuid** (str) – workflow uuid
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
The success flag.

---

**Example**

To map a workflow 'st1234' to finding by id '123' of type severitychange

```
>>> self.{risksenseobject}.host_findings.map_findings([{"field":"id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"123"}],'severityChange','st1234')
```

---

**Note:** You can also dump the host findings data post mapping the findings using csvdump=True argument

```
>>> self.{risksenseobject}.host_findings.map_findings([{"field":"id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"123"}],'severityChange','st1234',csvdump=True)
```

**add_ticket_tag**(*search_filters*, *tag_id*, *client_id=None*)

Adds a ticket tag to the host findings based on a search filter

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **tag_id** (int) – The tag id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform is returned.

**Example**

To add a ticket tag to host findings

```
>>> self.{risksenseobject}.host_findings.add_ticket_tag([],123)
```

**unmap_findings**(*filter_request*, *workflowtype*, *workflowuuid*, *csvdump=False*, *client_id=None*)

Unmap findings from workflow.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowtype** (str) – Type of workflow, either falsePositive,remediation,acceptance,severityChange. Please use the exact names as above for workflow type
- **workflowuuid** (str) – workflow uuid
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
The success flag.

**Example**

To unmap a workflow 'st1234' from finding by id '123' of type severitychange

```
>>> self.{risksenseobject}.host_findings.unmap_findings([{"field":"id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"123"}],'severityChange','st1234')
```

**Note:** You can also dump the host findings data before unmapping the findings using `csvdump=True` argument

```
>>> self.{risksenseobject}.host_findings.unmap_findings([{"field":"id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"123"}],'severityChange','st1234',csvdump=True)
```

**get_model**(*client_id=None*)

Get available projections and models for Host Findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Host Finding projections and models are returned.

**Example**

An example to use get_model is

```
>>> self.{risksenseobject}.host_findings.get_model()
```

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2
- **client_id** (typing.Optional[int]) – Client ID

**Returns**
Value suggestions

**Example**

To use suggest function is

```
>>> self.{risksenseobject}.host_findings.suggest([],{})
```

**apply_system_filters**(*csvdump=False*, *client_id=None*)

Get data from system filters for host findings.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

- **csvdump** (bool) – csvdump is a boolean which you can make true if you want to dump the data from system filters in a csv. Keep it false if it's not needed.

**Return type**
list

**Returns**
The data of the system filter based host findings values are returned

---

**Example**

An example to use apply_system_filters is

```
>>> self.{risksenseobject}.host_findings.apply_system_filters()
```

The system filters will be displayed in the terminal to which you must provide a key value and the data returned will reflect based on the system filter chosen

---

**Note:** You can also dump the host findings from the system filters search by csvdump=True argument

```
>>> self.{risksenseobject}.host_findings.apply_system_filters(csvdump=True)
```

---

### 1.2.11 Patch (risksense_api.__subject.__patch.__patch)

**Patch**

**class** risksense_api.__subject.__patch.__patch.**Patch**(*profile*)

Bases: Subject

**Class for Patch function defintions**.

To utlise Patch function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.patch.{function}

---

**Examples**

To search for patch using get_model() function

```
>>> self.{risksenseobject}.patch.search({filter})
```

---

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

**Exports and Downloads a file based on the filters defined** .

**Parameters**

- **filename** (str) – Name of the file to export as
- **filters** (list) – Patch search filters based on which the export performs

- **client_id** (typing.Optional[int]) – The client id to get the data from. If not supplied, takes default client id

**Examples**

```
>>> self.{risksenseobject}.patch.downloadfilterinexport('patchdata',[])
```

**__init__**(*profile*)

**Initialization of Patch Object** .

**Parameters**
**profile** (object) – Profile Object

**getexporttemplate**(*client_id=None*)

Gets configurable export template for patches.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.patch.getexporttemplate()
```

This gets all the export templates for patch

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*)

Initiates an export job on the platform for patche(s) based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Available options ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform from is returned.

**Example**

An example to use export is

```
>>> self.{risksenseobject}.patch.export([],'testingexport')
```

You can change the filetype to any of the names above or even the other positional arguments as mentioned

```
>>> self.{risksenseobject}.patch.export([],'testingexport',file_
↪type=ExportFileType.JSON)
```

**get_filter**(*client_id=None*)

Get a list of supported patch filters.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
Filter information

**Examples**

```
>>>  self.{risksenseobject}.patch.get_filter()
```

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns patch based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_num** (int) – The page number of results to be returned.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

**Example**

An example to get single search page of patch data

```
>>> self.[risksenseobject].patch.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.[risksenseobject].patch.get_single_search_page([],page_num=2,page_
↪size=10)
```

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns patch data based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all patch returned by the search using the filter provided.

**Example**

An example to search for patches data is

```
>>> self.[risksenseobject].patch.search([])
```

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.[risksenseobject].patch.search([],csvdump=True)
```

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
suggestions

---

**Example**

To use suggest function is

```
>>> self.{risksenseobject}.patch.suggest([],{})
```

---

### 1.2.12 Playbooks (risksense_api.__subject.__playbooks.__playbooks)

**Playbooks module defined for different playbooks related api endpoints.**

**class** risksense_api.__subject.__playbooks.__playbooks.**Playbooks**(*profile*)

Bases: Subject

Playbooks class

**Parameters**
**profile** (object) –

**__init__**(*profile*)

Initialization of Playbooks object.

**Parameters**
**profile** (object) – Profile Object

**get_supported_inputs**(*csvdump=False*, *client_id=None*)

Get a list of supported playbook inputs.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID
- **csvdump** (bool) – dumps the data in csv

**Return type**
list

**Returns**
Supported inputs

---

**Example**

To get supported inputs

```
>>> self.{risksenseobject}.playbooks.get_supported_inputs()
```

**Note:** You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.get_supported_inputs(csvdump=True)
```

**get_supported_actions**(*csvdump=False*, *client_id=None*)

Get a list of supported playbook actions.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID
- **csvdump** (bool) – Dumps the data in csv

**Return type**
list

**Returns**
Supported actions

**Example**

To get supported actions

```
>>> self.{risksenseobject}.playbooks.get_supported_actions()
```

**Note:** You can also dump the data in a csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.get_supported_actions(csvdump=True)
```

**get_supported_frequencies**(*client_id=None*)

Get a list of supported playbook frequencies.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID
- **csvdump** – Dumps the data in csv

**Return type**
list

**Returns**
Supported frequencies

**Example**

To get supported frequencies

```
>>> self.{risksenseobject}.playbooks.get_supported_frequencies()
```

**Note:** You can also dump the data in a csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.get_supported_frequencies(csvdump=True)
```

`get_supported_outputs`(*csvdump=False*, *client_id=None*)

Get a list of supported playbook outputs.

**Parameters**

- **client_id** (`typing.Optional[int]`) – Client ID
- **csvdump** (`bool`) – Dumps the data in csv

**Return type**
`list`

**Returns**
Supported outputs

**Example**

To get supported outputs

```
>>> self.{risksenseobject}.playbooks.get_supported_outputs()
```

**Note:** You can also dump the data in a csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.get_supported_outputs(csvdump=True)
```

`get_subject_supported_actions`(*csvdump=False*, *client_id=None*)

Get a list of subject-supported playbook actions.

**Parameters**

- **client_id** (`typing.Optional[int]`) – Client ID
- **csvdump** (`bool`) – Dumps the data in csv

**Return type**
`dict`

**Returns**
Subject Supported actions

**Example**

To get Subject Supported actions

```
>>> self.{risksenseobject}.playbooks.get_subject_supported_actions()
```

**Note:** You can also dump the data in a csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.get_subject_supported_actions(csvdump=True)
```

---

**get_playbooks_single_page**(*page_size=1000*, *page_num=0*, *sort_dir='ASC'*, *client_id=None*)

Fetch a single page of playbooks from client

**Parameters**

- **page_size** (int) – Page Size
- **page_num** (int) – Page Number
- **sort_dir** (str) – Sort Direction
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
The paginated JSON response from the platform is returned.

---

**Example**

An example to get single search page of playbooks data

```
>>> self.{risksenseobject}.playbooks.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.{risksenseobject}.playbooks.get_single_search_page([],page_num=2,page_
↪size=10)
```

---

**get_all_playbooks**(*csvdump=False*, *client_id=None*)

Get all playbooks for a client.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID
- **csvdump** (bool) – dumps the data in csv

**Return type**
list

**Returns**
All Playbooks for a client

---

**Example**

To get all playbooks

```
>>>  self.{risksenseobject}.playbooks.get_all_playbooks()
```

---

**Note:** You can also dump the data using csvdump=True argument

```
>>> self.{risksenseobject}.playbooks.get_all_playbooks(csvdump=True)
```

---

**get_specific_playbook**(*playbook_uuid*, *csvdump=False*, *client_id=None*)

Fetch a specific playbook by UUID.

**Parameters**

- **playbook_uuid** (str) – Playbook UUID
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
The Playbook information

---

**Example**

To get specific playbook *1234str*

```
>>> self.{risksenseobject}.playbooks.get_specific_playbook('1234str')
```

---

**Note:** You can also dump the data using csvdump=True argument

```
>>> self.{risksenseobject}.playbooks.get_specific_playbook('1234str',
↪csvdump=True)
```

---

**get_single_page_playbook_rules**(*playbook_uuid*, *page_num=0*, *page_size=1000*, *sort_dir='ASC'*, *client_id=None*)

Get a single page of rules for a specific playbook

**Parameters**

- **playbook_uuid** (str) – Playbook UUID
- **page_num** (int) – Page number to retrieve
- **page_size** (int) – Number of items per page to return
- **sort_dir** (int) – Sort Direction
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Playbook rules

---

**Example**

To get single page playbook rule from playbook *123str*

```
>>> self.{risksenseobject}.playbooks.get_single_page_playbook_rules('123str')
```

**get_all_rules_for_playbook**(*playbook_uuid*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Get all rules for a specific playbook

**Parameters**

- **playbook_uuid** (str) – Playbook UUID
- **sort_dir** (str) – Sort Direction
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
All playbook rules

**Example**

To get all rules for playbook *123str*

```
>>> self.{risksenseobject}.playbooks.get_all_rules_for_playbook('123str')
```

**Note:** You can also dump the data using csvdump=True argument

```
>>> self.{risksenseobject}.playbooks.get_all_rules_for_playbook('123str',
↪csvdump=True)
```

**add_rule**(*playbook_uuid*, *rule_name*, *rule_desc*, *rule_input*, *rule_action_type*, *rule_action*, *rule_output_type*, *rule_output*, *csvdump=False*, *client_id=None*)

Add a rule to a playbook.

**Parameters**

- **playbook_uuid** (str) – Playbook UUID
- **rule_name** (str) – Rule Name
- **rule_desc** (str) – Rule Description
- **rule_input** (str) – Rule Input
- **rule_action_type** (str) – Rule Action Type
- **rule_action** (dict) – Rule action to take
- **rule_output_type** (str) – Rule output type
- **rule_output** (dict) – Rule output
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
`list`

**Returns**
List containing dict of rule details.

---

**Example**

To add a rule to a playbook

```
>>> self.{risksenseobject}.playbooks.add_rule('11ec58c8-123-123-a0b0-
↪06933745a4d6','newtest',"testingsomethinghere","HOST_FINDING","ASSIGNMENT",{
↪"userIds":[123],"filterRequest":{"filters":[{"field":"group_names","exclusive
↪":False,"operator":"EXACT","value":"AdamM","orWithPrevious":False,"enabled
↪":True,"implicitFilters":[],"altQueryConstruction":False},{"field":
↪"lastFoundOn","exclusive":False,"operator":"BEFORE","value":"2021-01-28",
↪"orWithPrevious":False,"enabled":True,"implicitFilters":[],
↪"altQueryConstruction":False}]}},"NO_OUTPUT",{})
```

---

**Note:** You can also dump the data using `csvdump=True` argument

```
>>> self.{risksenseobject}.playbooks.add_rule('11ec58c8-123-123-a0b0-
↪06933745a4d6','newtest',"testingsomethinghere","HOST_FINDING","ASSIGNMENT",{
↪"userIds":[123],"filterRequest":{"filters":[{"field":"group_names","exclusive
↪":False,"operator":"EXACT","value":"AdamM","orWithPrevious":False,"enabled
↪":True,"implicitFilters":[],"altQueryConstruction":False},{"field":
↪"lastFoundOn","exclusive":False,"operator":"BEFORE","value":"2021-01-28",
↪"orWithPrevious":False,"enabled":True,"implicitFilters":[],
↪"altQueryConstruction":False}]}},"NO_OUTPUT",{},csvdump=True)
```

---

**add_multiple_rules**(*playbook_uuid*, *rules*, *csvdump=False*, *client_id=None*)

Add multiple rules to a playbook.

**Parameters**

- **playbook_uuid** (`str`) – Playbook UUID
- **rules** (`list`) – List of Rules the user want to create
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID

**Return type**
`list`

**Returns**
List containing dict of rule details.

---

**Example**

To add multiple rules for a playbook

```
>>> self.{risksenseobject}.playbooks.add_multiple_rules('11ec8a6e-1234-123-
↪9fb0-02a87de7e1ee',[
{"name": "testnew2", "description": "test", "input": "HOST", "actionType": "TAG_
↪APPLY", "action": {"tagIds": [], "isRemove": False, "filterRequest": {"filters
↪": [{"field": "criticality", "exclusive": False, "operator": "IN", "value": "4
↪", "orWithPrevious": False, "implicitFilters": [], "enabled": True}]}},
↪"outputType": "NO_OUTPUT", "output": {}},{"name": "testnew3", "description":
↪"testing2", "input": "HOST", "actionType": "TAG_APPLY", "action": {"tagIds":␣
↪[], "isRemove": False, "filterRequest": {"filters": [{"field": "criticality",
↪"exclusive": False, "operator": "IN", "value": "4", "orWithPrevious": False,
↪"implicitFilters": [], "enabled": True}]}}, "outputType": "NO_OUTPUT", "output
↪": {}}])
```

**Note:** You can also dump the data using `csvdump=True` argument

```
>>> self.{risksenseobject}.playbooks.add_multiple_rules('11ec8a6e-1234-123-
↪9fb0-02a87de7e1ee',[
{"name": "testnew2", "description": "test", "input": "HOST", "actionType": "TAG_
↪APPLY", "action": {"tagIds": [], "isRemove": False, "filterRequest": {"filters
↪": [{"field": "criticality", "exclusive": False, "operator": "IN", "value": "4
↪", "orWithPrevious": False, "implicitFilters": [], "enabled": True}]}},
↪"outputType": "NO_OUTPUT", "output": {}},{"name": "testnew3", "description":
↪"testing2", "input": "HOST", "actionType": "TAG_APPLY", "action": {"tagIds":␣
↪[], "isRemove": False, "filterRequest": {"filters": [{"field": "criticality",
↪"exclusive": False, "operator": "IN", "value": "4", "orWithPrevious": False,
↪"implicitFilters": [], "enabled": True}]}}, "outputType": "NO_OUTPUT", "output
↪": {}}],csvdump=True)
```

**create**(*name*, *description*, *schedule_freq*, *hour_of_day*, *client_id=None*, *csvdump=False*, ***kwargs*)

Create a new playbook

**Parameters**

- **name** (str) – Name
- **description** (str) – Description
- **schedule_freq** (str) – Schedule Frequency (ScheduleFreq.DAILY, ScheduleFreq.WEEKLY, ScheduleFreq.MONTHLY, 'DISABLED')
- **hour_of_day** (str) – Hour of the day
- **client_id** (typing.Optional[int]) – Client ID
- **csvdump** (bool) – dumps the data in csv

**Keyword Arguments**

- **day_of_week** (str) – Day of the week
- **day_of_month** (str) – Day of the month

**Return type**

str

**Returns**
Playbook UUID

---

**Example**

To create a playbook

```
>>> self.{risksenseobject}.playbooks.create("teamtest1","test",self.rs.
↪schedulefreq.DAILY,"5")
```

---

**Note:** You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.create("teamtest1","test",self.rs.
↪schedulefreq.DAILY,"5",csvdump=True)
```

---

**update**(*playbook_uuid*, *name*, *description*, *schedule_freq*, *hour_of_day*, *csvdump=False*, *client_id=None*, ***kwargs*)

Update a playbook

**Parameters**

- **playbook_uuid** (str) – Playbook UUID
- **name** (str) – Name
- **description** (str) – Description
- **schedule_freq** (str) – Schedule Frequency (ScheduleFreq.DAILY, ScheduleFreq.WEEKLY, ScheduleFreq.MONTHLY, 'DISABLED')
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID
- **hour_of_day** (str) – Hour of the day

**Keyword Arguments**

- **day_of_week** (str) – Day of the week
- **day_of_month** (str) – Day of the month

**Return type**
dict

**Returns**
Playbook and its details

---

**Example**

To update a playbook

```
>>> self.{risksenseobject}.playbooks.update('123456-3f1c-3b81-b7ab-06933745a4d6
↪','testing2','somethingtotestrighthere',"DAILY",hour_of_day=5)
```

---

**Note:** You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.update('123456-3f1c-3b81-b7ab-06933745a4d6
↪','testing2','somethingtotestrighthere',"DAILY",hour_of_day=5,csvdump=True)
```

**delete**(*playbook_uuid*, *csvdump=False*, *client_id=None*)

Delete a playbook.

**Parameters**

- **playbook_uuid** (str) – playbook UUID
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – client ID

**Return type**
bool

**Returns**
true/false indicating successful deletion

**Example**

To delete a playbook

```
>>> self.{risksenseobject}.playbooks.delete('123-123')
```

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.playbooks.delete('123-123',csvdump=True)
```

**get_playbook_details**(*playbook_uuid*, *csvdump=False*, *client_id=None*)

Get the details for a specific playbook

**Parameters**

- **playbook_uuid** (str) – playbook UUID
- **client_id** (typing.Optional[int]) – client ID
- **csvdump** (bool) – Dump the data in a csv

**Return type**
dict

**Returns**
Playbook details

**Example**

To get playbook details

```
>>> self.{risksenseobject}.get_playbook_details('123-123')
```

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.get_playbook_details('123-123',csvdump=True)
```

---

**rule_reorder**(*playbook_uuid*, *rule_uuids*, *csvdump=False*, *client_id=None*)

Reorder playbook rules for an already existing playbook

**Parameters**

- **playbook_uuid** (str) – UUID for playbook to be reordered
- **rule_uuids** (list) – A list of rule UUIDs (strings), in the order desired
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
List of reordered rule definitions

---

**Example**

To reorder the rules

```
>>> self.{risksenseobject}.playbooks.rule_reorder('1234-87dc-353b-a0b0-
↪06933745a4d6',["4321-10bc-3f1f-a0b0-06933745a4d6",'1234-1151-3d17-b7ab-
↪06933745a4d6',"111-55bc-421a-b7ab-06933745a4d6","111-28fa-b4eb-b7ab-
↪06933745a4d6","111-fa9b-e4ad-b7ab-06933745a4d6"])
```

---

**Note:** You can also dump the reodered data in a csv using

```
>>> self.{risksenseobject}.playbooks.rule_reorder('1234-87dc-353b-a0b0-
↪06933745a4d6',["4321-10bc-3f1f-a0b0-06933745a4d6",'1234-1151-3d17-b7ab-
↪06933745a4d6',"111-55bc-421a-b7ab-06933745a4d6","111-28fa-b4eb-b7ab-
↪06933745a4d6","111-fa9b-e4ad-b7ab-06933745a4d6"],csvdump=True)
```

---

**update_rule**(*rule_uuid*, *playbook_name*, *playbook_desc*, *playbook_input*, *playbook_action_type*, *playbook_action*, *playbook_output_type*, *playbook_output*, *csvdump=False*, *client_id=None*)

Update an existing playbook rule

**Parameters**

- **rule_uuid** (str) – UUID for rule to be updated
- **playbook_name** (str) – Playbook name
- **playbook_desc** (str) – Playbook description
- **playbook_input** (str) – Playbook Input
- **playbook_action_type** (str) – Playbook action type
- **playbook_action** (dict) – Playbook action
- **playbook_output_type** (str) – Playbook output type

- **playbook_output** (dict) – Playbook output
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
bool

**Returns**
Indication of success

---

**Example**

To update a playbook rule

```
>>> self.{risksenseobject}.playbooks.update_rule('11ec8ae5-73dd-c48c-9fb0-
↪02a87de7e1ee',"namingconventionchanged","testnew2", "HOST", "TAG_APPLY", {
↪"tagIds": [], "isRemove": False, "filterRequest": {"filters": [{"field":
↪"criticality", "exclusive": False, "operator": "IN", "value": "4",
↪"orWithPrevious": False, "implicitFilters": [], "enabled": True}]},"NO_OUTPUT
↪", {})
```

---

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.playbooks.update_rule('11ec8ae5-73dd-c48c-9fb0-
↪02a87de7e1ee',"namingconventionchanged","testnew2", "HOST", "TAG_APPLY", {
↪"tagIds": [], "isRemove": False, "filterRequest": {"filters": [{"field":
↪"criticality", "exclusive": False, "operator": "IN", "value": "4",
↪"orWithPrevious": False, "implicitFilters": [], "enabled": True}]},"NO_OUTPUT
↪", {},csvdump=True)
```

---

**delete_playbook_rule**(*rule_uuid*, *csvdump=False*, *client_id=None*)

Delete an existing playbook rule.

**Parameters**

- **rule_uuid** (str) – Rule UUID
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
bool

**Returns**
Indication of success

---

**Example**

To delete a playbook rule

```
>>> self.{risksenseobject}.playbooks.delete_playbook_rule('1234-6fb3-206e-9fb0-
↪02a87de7e1ee')
```

**Note:** You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.delete_playbook_rule('1234-6fb3-206e-9fb0-
↪02a87de7e1ee',csvdump=True)
```

`get_specific_playbook_rule`(*rule_uuid*, *csvdump=False*, *client_id=None*)

Get details for a specific playbook rule.

**Parameters**

- **rule_uuid** (`str`) – Playbook rule UUID
- **client_id** (`typing.Optional[int]`) – Client ID
- **csvdump** (`bool`) – dumps the data in csv

**Return type**
`dict`

**Returns**
Playbook rule details

**Example**

To get specific playbook rule

```
>>> self.{risksenseobject}.playbooks.get_specific_playbook_rule('123456-73dd-
↪c48c-9fb0-02a87de7e1ee')
```

**Note:** You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.playbooks.get_specific_playbook_rule('123456-73dd-
↪c48c-9fb0-02a87de7e1ee',csvdump=True)
```

`toggle_enabled`(*playbook_uuids*, *enabled=False*, *client_id=None*)

Enable/Disable playbooks.

**Parameters**

- **playbook_uuids** (`list`) – A list of Playbook UUIDs to enable/disable
- **enabled** (`bool`) – Enable/Disable playbooks,please provide true for enabled and false for disabled
- **client_id** (`typing.Optional[int]`) – Client ID

**Returns**
True

**Example**

To enable a playbook

```
>>> self.{risksenseobject}.playbooks.toggle_enabled(['11ed13b4-52c3-a3c1-9fb0-
↪02a87de7e1ee'],enabled=True)
```

To disable a playbook

```
>>> self.{risksenseobject}.playbooks.toggle_enabled(['11ed13b4-52c3-a3c1-9fb0-
↪02a87de7e1ee'],enabled=False)
```

---

**run_playbook**(*playbook_uuid*, *csvdump=False*, *client_id=None*)

Run a playbook.

**Parameters**

- **playbook_uuid** (str) – Playbook UUID
- **client_id** (typing.Optional[int]) – Client ID
- **csvdump** (bool) – dumps the data in csv

**Return type**
dict

**Returns**
JSON response from platform

---

**Example**

```
>>> self.{risksenseobject}.playbooks.run_playbook('12345-1234-123')
```

---

---

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.playbooks.run_playbook('12345-1234-123',csvdump=True)
```

---

**_get_playbook_page_info**(*url*, *page_size*)

Get number of available pages for fetch.

**Parameters**

- **url** (str) – URL of endpoint
- **page_size** (int) – page size

**Return type**
int

**Returns**
Total number of available pages

**IGNORE function as it is an Internal Function***

**_fetch_in_bulk**(*func_name*, *page_range*, ***func_args*)

Threaded fetch of playbook info, supporting multiple threads. Combines all results in a single list and returns.

**Parameters**

- **func_name** (str) – Search function name
- **page_range** (int) – Page range

*IGNORE - INTERNAL FUNCTION*

**Keyword Arguments**
**func_args** (dict) – args to be passed to search function

**Return type**
list

**Returns**
List of all results returned by search function

### 1.2.13 Rs3 (risksense_api.__subject.__rs3.__rs3)

**Rs3 module defined for different rs3 related api endpoints.**

**class** risksense_api.__subject.__rs3.__rs3.**Rs3**(*profile*)

Bases: Subject

**Class for Rs3 function defintions**.

To utlise rs3 function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.rs3.{function}

---

**Examples**

To get rs3 over time aggregate use *get_rs3overtimeaggregate()* function

```
>>> self.{risksenseobject}.rs3.get_rs3overtimeaggregate({args})
```

---

**__init__**(*profile*)

Initialization of Rs3 object.

**Profile**
Profile Object

**Parameters**
**profile** (object) –

**get_rs3overtimeaggregate**(*startdate*, *enddate*, *filters*, *csvdump=False*, *client_id=None*)

Gets rs3 aggregate score between dates

**Parameters**

- **startdate** (str) – The start date from when rs3 score is needed,please mention date
- **format** (*in YYY-MM-DD*) –
- **enddate** (str) – The end date till when rs3 score is needed,please mention date
- **format** –

- **filters** (list) – filters to define for the rs3
- **csvdump** (bool) – Dump the data in csv
- **client_id** (*type*) – client id , if none takes default client_id
- **client_id** – int

**Return type**
dict

**Returns**
The jsonified response from the platform

---

**Example**

To get rs3 overtime aggregate

```
>>> self.rs.rs3.get_rs3overtimeaggregate('2022-02-11','2022-03-11')
```

---

**Note:** To dump the data in csv, you can use `csvdump=True` argument

```
>>> self.rs.rs3.get_rs3overtimeaggregate('2022-02-11','2022-03-11',csvdump=True)
```

---

**get_rs3aggregate**(*search_filter*, *applymecheck=True*, *csvdump=False*, *client_id=None*)

Gets rs3 aggregate score

**Parameters**

- **search_filter** (list) – Search filters for rs3 aggregate
- **applymecheck** (bool) – Apply manual exploit check for client rs3 with default value true
- **csvdump** (bool) – Dump the data in csv
- **client_id** (typing.Optional[int]) – client id , if none takes default client_id

**Return type**
dict

**Returns**
The rs3 aggregate data

---

**Example**

```
>>> self.rs.rs3.get_rs3aggregate([])
```

---

**Note:** To dump the data in csv, you can use `csvdump=True` argument

```
>>> self.rs.rs3.get_rs3aggregate([],csvdump=True)
```

---

**get_rs3historyaggregate**(*startdate*, *enddate*, *search_filter*, *csvdump=False*, *client_id=None*)

Gets rs3 aggregate history between dates

**Parameters**

- **startdate** (str) – The start date from when rs3 score is needed
- **enddate** (str) – The end date till when rs3 score is needed
- **filters** – filters to define for the rs3
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – client id , if none takes default client_id

**Return type**
list

**Returns**
The rs3 history

---

**Example**

To get rs3 history aggregate

```
>>> self.rs.rs3.get_rs3historyaggregate('2022-02-11','2022-03-11',[])
```

---

**Note:** To dump the data in csv, you can use `csvdump=True` argument

```
>>> self.rs.rs3.get_rs3historyaggregate('2022-02-11','2022-03-11',[],
↪csvdump=True)
```

---

**Parameters**
**search_filter** (list) –

**simulate_rs3**(*vrrCriticalMax*, *vrrHighMax*, *vrrMediumMax*, *vrrLowMax*, *findingCount*, *assetType*, *assetCriticality*, *assetCategory*, *client_id=None*)

Simulate rs3 score based on the vrr,findingcount,asset data

**Parameters**

- **vrrCriticalMax** (float) – The vrrCriticalMax info
- **vrrHighMax** (float) – The vrrhighmax info
- **vrrMediumMax** (float) – The vrrMediumMax info
- **vrrLowMax** (float) – The vrrLowMax info
- **findingCount** (int) – The number of findings
- **assetType** (str) – The type of asset either external or internal
- **assetCriticality** (int) – The asset criticality
- **assetCategory** (str) – The asset category
- **client_id** (typing.Optional[int]) – client id , if none takes default client_id

**Return type**
int

**Returns**
The rs3 simulated information

---

**Example**

To simulate the rs3 with asset category host and asset type external with our criticality

```
>>> self.rs.rs3.simulate_rs3(9.1 7.1 5.1 2.1 4 'External' 3 'Host')
```

---

### 1.2.14 Quickfilters (risksense_api.__subject.__quickfilters.__quickfilters)

**Quick filters module defined for different quick filters related api endpoints.**

**class** risksense_api.__subject.__quickfilters.__quickfilters.**Quickfilters**(*profile*)

Bases: Subject

**Class for Quickfilter function defintions**.

To utlise Quickfilter function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**
self.{risksenseobjectname}.quickfilters.{function}

---

**Examples**

To get weakness quickfilters using *get_weakness_quickfilters()* function

```
>>> self.rs.quickfilters.get_weakness_quickfilters([])
```

---

**__init__**(*profile*)

Initialization of quickfilters object.

profile: Profile Object :type profile: _profile

**Parameters**
**profile** (object) –

**get_vulnerability_quickfilters**(*quickfilters*, *clientid=None*)

Get vulnerability quickfilters based on filters in the search endpoint

**Parameters**

- **quickfilters** (list) – Filters that need to get quick filters
- **clientid** (typing.Optional[int]) – Client id , if none for default client id

**Returns**
The jsonified response from the platform

**Return type**
jsonified_response

**Example**

```
>>> self.rs.quickfilters.get_vulnerability_quickfilters([])
```

**get_weakness_quickfilters**(*quickfilters*, *clientid=None*)

Get weakness quickfilters based on filters in the search endpoint

**Parameters**

- **quickfilters** (list) – Filters that need to get quick filters
- **clientid** (typing.Optional[int]) – Client id , if none for default client id

**Return type**
dict

**Returns**
Jsonified_response

**Example**

```
>>> self.rs.quickfilters.get_weakness_quickfilters([])
```

### 1.2.15 Sla (risksense_api.__subject.__sla.__sla)

**Sla module defined for different sla related api endpoints.**

**class** risksense_api.__subject.__sla.__sla.**SlaActionType**

Bases: object

Types of sla action type

**REMEDIATION_SLA = 'REMEDIATION_SLA'**

**class** risksense_api.__subject.__sla.__sla.**SlaMatrix**

Bases: object

Types of Sla Matrix

**STANDARD = {'1': [45, 90, 90, 120, 0], '2': [30, 90, 90, 120, 0], '3': [21, 45, 90, 120, 0], '4': [14, 30, 90, 120, 0], '5': [7, 21, 90, 120, 0]}**

**ACCELERATED = {'1': [30, 60, 90, 90, 0], '2': [21, 45, 90, 90, 0], '3': [14, 30, 60, 90, 0], '4': [7, 21, 45, 90, 0], '5': [3, 14, 30, 90, 0]}**

**AGGRESSIVE = {'1': [14, 30, 45, 60, 0], '2': [7, 21, 30, 60, 0], '3': [3, 14, 30, 60, 0], '4': [2, 7, 15, 60, 0], '5': [1, 3, 15, 60, 0]}**

**class** risksense_api.__subject.__sla.__sla.**SlaDataOperator**

Bases: object

Types of sla data operator

**MET_SLA = 'MET_SLA'**

**OVERDUE = 'OVERDUE'**

**MISSED_SLA = 'MISSED_SLA'**

**WITHIN_SLA = 'WITHIN_SLA'**

class risksense_api.__subject.__sla.__sla.**SlaMatrixProfileType**

Bases: object

Types of Sla Matrix Profile Type

**STANDARD = 'STANDARD'**

**AGGRESSIVE = 'AGGRESSIVE'**

**ACCELERATED = 'ACCELERATED'**

**CUSTOM = 'CUSTOM'**

class risksense_api.__subject.__sla.__sla.**TimeReference**

Bases: object

Types of sla time reference

**INGESTION_DATE = 'INGESTION_DATE'**

**DISCOVERED_DATE = 'DISCOVERED_DATE'**

class risksense_api.__subject.__sla.__sla.**offsetbasis**

Bases: object

Types of sla offset basis

**VRR = 'VRR'**

**SEVERITY = 'SEVERITY'**

class risksense_api.__subject.__sla.__sla.**Sla**(*profile*)

Bases: Subject

Sla class

**Parameters**
**profile** (object) –

**__init__**(*profile*)

Initialization of sla object.

**Parameters**
**profile** (object) – Profile Object

**create_sla**(*description*, *schedule_type='DAILY'*, *hourofday=12*, *name='Remediation SLAs'*, *csvdump=False*, *client_id=None*, ***kwargs*)

Creates an sla

**Parameters**

- **name** (str) – Name of playbook,note for system sla please mention Remediation SLAs
- **description** (str) – Provide description for sla
- **schedule_type** (str) – Schedule Frequency (ScheduleFreq.DAILY, ScheduleFreq.WEEKLY, ScheduleFreq.MONTHLY, 'DISABLED')

- **hourofday** (int) – Hour of the day
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID
- **Args** (*Keyword*) – day_of_week(str): Day of the week day_of_month(str): Day of the month

**Return type**
str

**Returns**
Playbook UUID

---

**Example**

To create a remediation sla

```
>>> self.{risksenseobject}.sla.create_sla('remediation sla')
```

---

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.sla.create_sla('remediation sla',csvdump=True)
```

---

**getslarules**(*playbookuuid*, *csvdump=False*, *client_id=None*)

Get a list of all sla rules for an sla.

**Parameters**

- **playbookuuid** (str) – Uuid of the playbook to fetch sla
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
list

**Returns**
Sla rules

---

**Example**

To get list of all sla rules

```
>>> self.{risksenseobject}.sla.getslarules('11ed05cb-e7a0-4f25-b7ab-06933745a4d6
↪')
```

---

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.sla.getslarules('11ed05cb-e7a0-4f25-b7ab-06933745a4d6
↪',csvdump=True)
```

**getallslas**(*csvdump=False*, *client_id=None*)

Gets all slas for a particular client.

**Parameters**

- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
dict

**Returns**
To get all slas

**Example**

To get all slas

```
>>> self.{risksenseobject}.getallslas()
```

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.getallslas(csvdump=True)
```

**delete_sla**(*sla_uuid*, *csvdump=False*, *client_id=None*)

Delete a particular sla

**Parameters**

- **sla_uuid** (str) – UUID of sla
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
bool

**Returns**
Success

**Example**

```
>>> self.{risksenseobject}.sla.delete_sla('1234-123-9f18-b7ab-06933745a4d6')
```

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.sla.delete_sla('1234-123-9f18-b7ab-06933745a4d6')
```

**get_sla_rule**(*playbookrulepairinguuid*, *csvdump=False*, *client_id=None*)

Gets all sla rules for a particular client.

**Parameters**

- **playbookrulepairinguuid** (str) – Playbook rule pairing uuid
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
dict

**Returns**
SLA details

---

**Example**

To get all sla rules

```
>>> self.{risksenseobject}.sla.get_sla_rule('12334-123-3bb6-9564-dbbe539b1a74')
```

---

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.sla.get_sla_rule('187914ba-b06e-3bb6-9564-
↪dbbe539b1a74',csvdump=True)
```

---

**get_specified_sla**(*playbookuuid*, *csvdump=False*, *client_id=None*)

Gets a specified sla for a particular client.

**Parameters**

- **playbookuuid** (str) – UUID of playbook
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
dict

**Returns**
Specific sla data

---

**Example**

To get specific sla

```
>>> self.{risksenseobject}.sla.get_specified_sla('12345st-123-4f25-b7ab-
↪06933745a4d6')
```

---

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.sla.get_specified_sla('11ed05cb-e7a0-4f25-b7ab-
↪06933745a4d6',csvdump=True)
```

---

**get_sla_details**(*playbookuuid*, *csvdump=False*, *client_id=None*)

Get details of a particular sla playbook.

**Parameters**

- **playbookuuid** (str) – SLA Playbook uuid
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
dict

**Returns**
Returns the sla playbook details

---

**Example**

To get sla details

```
>>> self.{risksenseobject}.sla.get_sla_details('11ed05cb-e7a0-4f25-b7ab-
↪06933745a4d6')
```

---

---

**Note:** You can also dump the data in csv using csvdump=True

```
>>> self.{risksenseobject}.sla.get_sla_details('11ed05cb-e7a0-4f25-b7ab-
↪06933745a4d6',csvdump=True)
```

---

**add_default_sla_rule**(*sla_uuid*, *description*, *priority*, *timeReference='DISCOVERED_DATE'*, *serviceLevelAgreementMatrix={'1': [45, 90, 90, 120, 0], '2': [30, 90, 90, 120, 0], '3': [21, 45, 90, 120, 0], '4': [14, 30, 90, 120, 0], '5': [7, 21, 90, 120, 0]}*, *slaMatrixProfileType='STANDARD'*, *offsetBasis='VRR'*, *affectOnlyNewFindings=True*, *updateSLAIfVRRUpdates=True*, *inputdata='HOST_FINDING'*, *actionType='REMEDIATION_SLA'*, *csvdump=False*, *client_id=None*)

Adds default rule to existing sla playbook.Works only if there is no default rule applied to the playbook

**Parameters**

- **sla_uuid** (str) – Sla UUID
- **description** (str) – Provide description for the default sla rule,
- **priority** (int) – Priority to be provided to default sla rule , note: default sla priority should be a greater number than the group sla rules
- **timeReference** (str) – Timereference to be provided to default sla rule ,
- **serviceLevelAgreementMatrix** (str) – Provide slamatrix for the particular sla
- **slaMatrixProfileType** (str) – Provide sla matrix type for the sla

- **offsetBasis** (str) – Provide offset basis for particular sla
- **affectOnlyNewFindings** (bool) – Choose whether it affects new findings
- **updateSLAIfVRRUpdates** (bool) – Choose if slaupdates with vrr
- **inputdata** (str) – Type of data provided for sla
- **actionType** (str) – Type of action for particular sla
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID , if no client id provided , takes default client id
- **action_type** – Action type for the particular sla rule,by default remediation rule is provided

**Return type**
list

**Returns**
List containing dict of rule details.

---

**Example**

To add default sla rule

```
>>> self.{risksenseobject}.add_default_sla_rule('11ed13e6-a1aa-5c28-9fb0-
↪02a87de7e1ee','testing',3)
```

---

**Note:** You can also change the default time Reference or service Level Agreement Matrix or other default arguments

You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.add_default_sla_rule('11ed13e6-a1aa-5c28-9fb0-
↪02a87de7e1ee','testing',3,csvdump=True)
```

---

**update_default_sla_rule**(*playbookrulepairinguuid*, *description*, *priority*, *timeReference='DISCOVERED_DATE'*, *serviceLevelAgreementMatrix={'1': [45, 90, 90, 120, 0], '2': [30, 90, 90, 120, 0], '3': [21, 45, 90, 120, 0], '4': [14, 30, 90, 120, 0], '5': [7, 21, 90, 120, 0]}*, *slaMatrixProfileType='STANDARD'*, *offsetBasis='VRR'*, *affectOnlyNewFindings=True*, *updateSLAIfVRRUpdates=True*, *inputdata='HOST_FINDING'*, *actionType='REMEDIATION_SLA'*, *csvdump=False*, *client_id=None*)

Updates default rule to existing sla playbook.

**Parameters**

- **playbookrulepairinguuid** (str) – Playbook rule pairing uuid
- **description** (str) – Provide description for the default sla rule,
- **priority** (int) – Priority to be provided to default sla rule , note: default sla priority should be a greater number than the group sla rules
- **timeReference** (str) – Timereference to be provided to default sla rule ,

- **serviceLevelAgreementMatrix** (dict) – Provide slamatrix for the particular sla
- **slaMatrixProfileType** (str) – Provide sla matrix type for the sla
- **offsetBasis** (str) – Provide offset basis for particular sla
- **affectOnlyNewFindings** (bool) – Choose whether it affects new findings
- **updateSLAIfVRRUpdates** (bool) – Choose if slaupdates with vrr
- **inputdata** (str) – Type of data provided for sla
- **actionType** (str) – Type of action for particular sla
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID , if no client id provided , takes default client id

**Return type**
list

**Returns**
List containing dict of rule details.

---

**Example**

To update default sla rule

```
>>> self.{risksenseobject}.update_default_sla_rule('11ed13e6-a1aa-5c28-9fb0-
↪02a87de7e1ee','testing',3)
```

---

**Note:** You can also change the default time Reference or service Level Agreement Matrix or other default arguments

You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.update_default_sla_rule('11ed13e6-a1aa-5c28-9fb0-
↪02a87de7e1ee','testing',3,csvdump=True)
```

---

**add_group_sla_rule**(*sla_uuid*, *name*, *description*, *priority*, *targetgroupids*, *timeReference='DISCOVERED_DATE'*, *serviceLevelAgreementMatrix={'1': [45, 90, 90, 120, 0], '2': [30, 90, 90, 120, 0], '3': [21, 45, 90, 120, 0], '4': [14, 30, 90, 120, 0], '5': [7, 21, 90, 120, 0]}*, *slaMatrixProfileType='STANDARD'*, *offsetBasis='VRR'*, *affectOnlyNewFindings=True*, *updateSLAIfVRRUpdates=True*, *inputdata='HOST_FINDING'*, *actionType='REMEDIATION_SLA'*, *csvdump=False*, *client_id=None*)

Adds group rule to existing sla playbook for the groups specified.

**Parameters**

- **sla_uuid** (str) – Sla UUID
- **name** (str) – Name of the group specific sla rule
- **description** (str) – Provide description for the group specific sla rule,
- **priority** (int) – Priority to be provided to group speicifc sla , note: group sla priority should be a lesser number than the default sla rule

- **targetgroupids** (list) – The groups ids where the sla applies
- **timeReference** (str) – Timereference to be provided to default sla rule ,
- **serviceLevelAgreementMatrix** (dict) – Provide slamatrix for the particular sla
- **slaMatrixProfileType** (str) – Provide sla matrix type for the sla
- **offsetBasis** (str) – Provide offset basis for particular sla
- **affectOnlyNewFindings** (bool) – Choose whether it affects new findings
- **updateSLAIfVRRUpdates** (bool) – Choose if slaupdates with vrr
- **inputdata** (str) – Type of data provided for sla
- **actionType** (str) – Type of action for particular sla
- **csvdump** (bool) – Dumps data to csv
- **client_id** (typing.Optional[int]) – Client ID , if no client id provided , takes default client id

**Return type**
list

**Returns**
List containing dict of rule details.

**Example**

```
>>> self.{risksenseobject}.sla.add_group_sla_rule(sla_uuid='12345st-123-5c28-
↪9fb0-02a87de7e1ee',name='testingnew',description='testingnew',
↪targetgroupids=[],priority=2)
```

**Note:** You can also change the default time Reference or service Level Agreement Matrix or other default arguments

You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.sla.add_group_sla_rule(sla_uuid='12345st-123-5c28-
↪9fb0-02a87de7e1ee',name='testingnew',description='testingnew',
↪targetgroupids=[],priority=2,csvdump=True)
```

**update_group_sla_rule**(*playbookrulepairinguuid*, *name*, *description*, *priority*, *targetgroupids*, *timeReference='DISCOVERED_DATE'*, *serviceLevelAgreementMatrix={'1': [45, 90, 90, 120, 0], '2': [30, 90, 90, 120, 0], '3': [21, 45, 90, 120, 0], '4': [14, 30, 90, 120, 0], '5': [7, 21, 90, 120, 0]}*, *slaMatrixProfileType='STANDARD'*, *offsetBasis='VRR'*, *affectOnlyNewFindings=True*, *updateSLAIfVRRUpdates=True*, *inputdata='HOST_FINDING'*, *actionType='REMEDIATION_SLA'*, *csvdump=False*, *client_id=None*)

Updates a group sla rule for an already existing playbook

**Parameters**

- **playbookrulepairinguuid** (str) – Playbook rule pairing uuid
- **name** (str) – Name of the group specific sla rule

- **description** (str) – Provide description for the group specific sla rule,
- **priority** (int) – Priority to be provided to group speicifc sla , note: group sla priority should be a lesser number than the default sla rule
- **targetgroupids** (list) – The groups ids where the sla applies
- **timeReference** (str) – Timereference to be provided to default sla rule ,
- **serviceLevelAgreementMatrix** (dict) – Provide slamatrix for the particular sla
- **slaMatrixProfileType** (str) – Provide sla matrix type for the sla
- **offsetBasis** (str) – Provide offset basis for particular sla
- **affectOnlyNewFindings** (bool) – Choose whether it affects new findings
- **updateSLAIfVRRUpdates** (bool) – Choose if slaupdates with vrr
- **inputdata** (str) – Type of data provided for sla
- **actionType** (str) – Type of action for particular sla
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID , if no client id provided , takes default client id
- **action_type** – Action type for the particular sla rule,by default remediation rule is provided

**Return type**
list

**Returns**
List containing dict of rule details.

---

**Example**

To update group sla rule

```
>>> self.[risksenseobject].sla.update_group_sla_rule('12345st-123-5c28-9fb0-
↪02a87de7e1ee','testingnew','testingnew',[],2)
```

---

**Note:** You can also change the default time Reference or service Level Agreement Matrix or other default arguments

You can also dump the data in csv using `csvdump=True`

```
>>> self.[risksenseobject].sla.update_group_sla_rule('12345st-123-5c28-9fb0-
↪02a87de7e1ee','testingnew','testingnew',[],2,csvdump=True)
```

---

**del_group_sla_rule**(*playbookuuid*, *client_id=None*)

Deletes a particular group sla rule.

**Parameters**

- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id
- **playbookuuid** (str) – The playbookuuid that needs to be deleted

**Return type**
`bool`

**Returns**
Returns json of deleted rule

---

**Example**

Delete group sla rule

```
>>> self.{risksenseobject}.sla.del_group_sla_rule('123-123')
```

---

**change_order**(*slauuid*, *ruleuuids*, *csvdump=False*, *client_id=None*)

Changes the order of the sla rules

**Parameters**

- **client_id** (`typing.Optional[int]`) – Client ID, if no client id provided , gets default client id
- **slauuid** (`str`) – The sla where rules needs to be reordered
- **ruleuuids** (`list`) – The order of rules reordering
- **csvdump** (`bool`) – dumps the data in csv

**Return type**
`list`

**Returns**
Returns the reordered rule json

---

**Example**

```
>>> self.rs.sla.change_order('123-4567-f4cb-b7ab-06933745a4d6',["97abc-123-
↪3bc0-b606-98aed43c944a"])
```

---

**Note:** You can also dump the data in csv using `csvdump=True`

```
>>> self.rs.sla.change_order('123-4567-f4cb-b7ab-06933745a4d6',["97abc-123-
↪3bc0-b606-98aed43c944a"],csvdump=True)
```

---

**sla_run**(*slauuid*, *csvdump=False*, *client_id=None*)

Runs the sla

**Parameters**

- **client_id** (`typing.Optional[int]`) – Client ID, if no client id provided , gets default client id
- **slauuid** (`str`) – The sla rule that needs to be run
- **csvdump** (`bool`) – Dumps the data in csv

**Return type**
`dict`

**Returns**
Returns the json of sla

**Example**

```
>>> self.{risksenseobject}.sla.sla_run('12345-123ea-4f25-b7ab-06933745a4d6')
```

**update_sla**(*sla_uuid*, *description*, *schedule_type='DAILY'*, *hourofday=0*, *dayofmonth='4'*, *dayofweek='5'*, *name='Remediation SLAs'*, *type='System'*, *csvdump=False*, *client_id=None*, ***kwargs*)

Updates an sla

**Parameters**

- **sla_uuid** (str) – Sla UUID
- **name** (str) – Name of the
- **description** (str) – Provide description for sla
- **schedule_type** (str) – Schedule Frequency (ScheduleFreq.DAILY, ScheduleFreq.WEEKLY, ScheduleFreq.MONTHLY, 'DISABLED')
- **type** (str) – The type of sla , can be 'SYSTEM' or 'USER'
- **hourofday** (int) – Hour of the day
- **dayofmonth** (str) – Day of the month
- **dayofweek** (str) – Day of the week
- **csvdump** (bool) – dumps the data in csv
- **client_id** – Client ID , if no client id provided , takes default client id

**Returns**
List containing dict of playbook details.

**Example**

```
>>> self.{risksenseobject}.sla.update_sla('1234-1234abc-bfc0-b7ab-06933745a4d6',
↪'remediation sla','Testingviascript')
```

**Note:** You can also dump the data in csv using `csvdump=True`

```
>>> self.{risksenseobject}.sla.update_sla('1234-1234abc-bfc0-b7ab-06933745a4d6',
↪'remediation sla','Testingviascript',csvdump=True)
```

**disablesla**(*playbookuuid*, *client_id=None*)

Disables a particular sla rule.

**Parameters**

- **playbookuuid** (str) – The playbookuuids that needs to be disabled
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
bool

**Returns**
Returns json of disable sla

**Example**

```
>>> self.{risksenseobject}.sla.disablesla(['11ed05cb-e7a0-4f25-b7ab-06933745a4d6
↪'])
```

**enablesla**(*playbookuuid*, *client_id=None*)

Enables a particular sla rule.

**Parameters**

- **playbookuuid** (list) – The playbookuuids that needs to be enabled
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
bool

**Returns**
success

**Example**

```
>>> self.{risksenseobject}.sla.enablesla(['11ed05cb-e7a0-4f25-b7ab-06933745a4d6
↪'])
```

**getplaybookslarules**(*slaid*, *client_id=None*)

Gets sla playbook rules for a particular sla.

**Parameters**

- **slaid** (int) – Slaid to get the sla rules
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
dict

**Returns**
Returns json of playbook rules

**Example**

To get sla playbook rule

```
>>> self.{risksenseobject}.sla.getplaybookslarules('11ed13e6-a1aa-5c28-9fb0-
↪02a87de7e1ee')
```

**delete_sla_rule**(*playbookrulepairinguuid*, *client_id=None*)

Deletes a particular sla rule

**Parameters**

- **playbookrulepairinguuid** (str) – the playbook rule uuid
- **client_id** (typing.Optional[int]) – Client ID, if no client id provided , gets default client id

**Return type**
bool

**Returns**
Success

**Example**

To delete an sla rule

```
>>> self.{risksenseobject}.sla.delete_sla_rule('94945073-3d62-35b6-b176-
↪6412bd324b84')
```

### 1.2.16 Tags (risksense_api.__subject.__tags.__tags)

**Tags module defined for different tags related api endpoints.**

**class** risksense_api.__subject.__tags.__tags.**TagType**

Bases: object

TagType class and attributes

**COMPLIANCE = 'COMPLIANCE'**

**LOCATION = 'LOCATION'**

**CUSTOM = 'CUSTOM'**

**REMEDIATION = 'REMEDIATION'**

**PEOPLE = 'PEOPLE'**

**PROJECT = 'PROJECT'**

**SCANNER = 'SCANNER'**

**CMDB = 'CMDB'**

**class** risksense_api.__subject.__tags.__tags.**Tags**(*profile*)

Bases: Subject

**Class for Tags function defintions**.

To utlise Tags function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**

```
self.{risksenseobjectname}.Tags.{function}
```

---

**Examples**

To search for tags using *search()* function

```
>>> self.{risksenseobject}.tags.search({filterobject})
```

---

**__init__**(*profile*)

Initialization of Tags object.

profile: Profile Object :type profile: _profile

**Parameters**

**profile** (object) –

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

**Exports and Downloads a file based on the filters defined** .

**Parameters**

- **filename** (str) – Name of the file to export as
- **filters** (list) – Tag search filters based on which the export performs
- **client_id** (typing.Optional[int]) – The client id to get the data from. If not supplied, takes default client id

**IGNORE INTERNAL FUNCTION**

---

**Examples**

```
>>> self.{risksenseobject}.tags.downloadfilterinexport('applicationfindingsdata
↪',[])
```

---

**create**(*tag_type*, *name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *propagate=True*, *csvdump=False*, *client_id=None*)

Create a new tag for the client.

**Parameters**

- **tag_type** (str) – Type of tag to be created.TagType.COMPLIANCE, TagType.LOCATION, TagType.CUSTOM, TagType.REMEDIATION, TagType.PEOPLE, TagType.PROJECT, TagType.SCANNER, TagType.CMDB
- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **propagate** (bool) – Propagate tag to all findings?

- **csvdump** (bool) – dumps the data in csv
- **client_id** – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The new tag ID will be returned.

---

**Example**

To create a tag of type people and give it a name rspackagetest and assign to user id 123

```
>>> self.rs.tags.create("PEOPLE",'rspackagetest','none',123)
```

---

**Note:** You can also dump the tag id created in a csv using `csvdump=True`:

```
>>> self.rs.tags.create("PEOPLE",'rspackagetest','none',123,csvdump=True)
```

For type specific tag creation , please view private functions section way below

---

**getexporttemplate**(*client_id=None*)

Gets configurable export template for Tags.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

---

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.tag.getexporttemplate()
```

This gets all the export templates for tags

---

**list_tag_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The JSON output from the platform is returned, listing the available filters.

**Examples**

```
>>>  self.{risksenseobject}.tags.list_tag_filter_fields()
```

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*)

Initiates an export job on the platform for Tag(s) based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Available options ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **exportable_filter** – Exportable filter
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform from is returned.

**Example**

An example to use export is

```
>>> self.{risksenseobject}.tags.export([],'testingexport')
```

You can change the filetype to any of the names above or even the other positional arguments as mentioned

```
>>> self.{risksenseobject}.tags.export([],'testingexport',file_
↪type=ExportFileType.JSON)
```

**update**(*tag_id*, *tag_type*, *name*, *desc*, *owner*, *color*, *locked*, *propagate=True*, *csvdump=False*, *client_id=None*)

Update an existing tag.

**Parameters**

- **tag_id** (int) – The tag ID to be updated.
- **tag_type** (str) – The type of tag.
- **name** (str) – The name of the tag.
- **desc** (str) – A description for the tag.
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"

- **color** (str) – The color for the tag. A hex value.
- **locked** (bool) – Whether or not the tag should be locked.
- **propagate** (bool) – Propagate tag to all findings?
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID will be returned.

---

**Example**

To update a tag id 123 of type people and give it a name rspackagetest and assign to user id 123

```
>>> self.rs.tags.update(123,"PEOPLE",'rspackagetest','none',123)
```

---

**Note:** You can also dump the tag based data in a csv using `csvdump=True`:

```
>>> self.rs.tags.update(123,"PEOPLE",'rspackagetest','none',123,csvdump=True)
```

For lock unlock specific tag update, please view private functions way below

---

**delete**(*tag_id*, *force_delete=True*, *csvdump=False*, *client_id=None*)

Delete a tag.

**Parameters**

- **tag_id** (int) – Tag ID to delete.
- **force_delete** (bool) – Indicates whether or not deletion should be forced.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Boolean reflecting the indication from the platform as to whether or not the deletion was successful.

---

**Example**

To delete a tag of id 123

```
>>> self.rs.tags.delete(123)
```

You can also dump the tag based data in a csv using `csvdump=True` argument:

```
>>> self.rs.tags.delete(269662,csvdump=True)
```

---

**bulk_tag_delete**(*search_filters*, *force_delete=True*, *csvdump=False*, *client_id=None*)

Delete a bunch of tags.

**Parameters**

- **search_filters** (list) – Tag ID to delete.
- **force_delete** (bool) – Indicates whether or not deletion should be forced.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**

**Boolean reflecting the indication from the platform as to whether**
or not the deletion was successful.

---

**Example**

To perform bulk tag delete

```
>>> self.rs.tags.bulk_tag_delete([])
```

---

**Note:** You can also dump the tags that are going to be deleted using csvdump=True argument:

```
>>> self.rs.tags.bulk_tag_delete([],csvdump=True)
```

---

**get_history**(*tag_id*, *page_num=0*, *page_size=20*, *csvdump=False*, *client_id=None*)

Get the history for a tag.

**Parameters**

- **tag_id** (int) – Tag ID
- **page_num** (int) – Page number to retrieve.
- **page_size** (int) – Number of items to be returned per page
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
A paginated JSON response from the platform is returned.

**Example**

To get history of the tag 123

```
>>> self.rs.tags.get_history(123)
```

**Note:** You can also dump the tag history by id created in a csv using `csvdump=True`:

```
>>> self.rs.tags.get_history(123,csvdump=True)
```

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns tags based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **page_num** (`int`) – Page number of results to be returned.
- **page_size** (`int`) – Number of results to be returned per page.
- **projection** (`str`) – Projection to use for query. Default is "basic"
- **sort_field** (`str`) – Name of field to sort results on.
- **sort_dir** (`str`) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`dict`

**Returns**
A paginated JSON response from the platform is returned.

**Example**

An example to get single search page of tags data

```
>>> self.{risksenseobject}.tags.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.{risksenseobject}.tags.get_single_search_page([],page_num=2,page_
↪size=10)
```

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns tags based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** – Projection to be used in API request. “basic” or “detail”
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (int) – Name of field to sort results on.
- **sort_dir** (int) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all tags returned by the search using the filter provided.

---

**Example**

An example to search for tags data is

```
>>> self.{risksenseobject}.tags.search([])
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.{risksenseobject}.tags.search([],csvdump=True)
```

---

**get_model**(*client_id=None*)

Get available projections and models for Tags.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Tags projections and models are returned.

---

**Example**

An example to use get_model is

```
>>> self.{risksenseobject}.tags.get_model()
```

---

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2

- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
Value suggestions

---

**Example**

To use suggest function is

```
>>> self.{risksenseobject}.tags.suggest([],{})
```

---

**create_compliance_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new COMPLIANCE tag.

**Parameters**

- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The new tag ID will be returned.

---

**Example**

To create a compliance tag 'testing' to user 123

```
>>> self.rs.tags.create_compliance_tag('testing','something',123,"#648d9f",True)
```

---

**Note:** You can also dump the tag id in a csv using the csvdump=True argument

```
>>> self.rs.tags.create_compliance_tag('testing','something',123,"#648d9f",True
↪csvdump=True)
```

---

**create_location_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new LOCATION tag.

**Parameters**

- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The new tag ID will be returned.

---

**Example**

To create a location tag 'testing' to user 123

```
>>> self.rs.tags.create_location_tag('testing','something',123,"#648d9f",True)
```

---

**Note:** You can also dump the tag id in a csv using the csvdump=True argument

```
>>> self.rs.tags.create_location_tag('testing','something',123,"#648d9f",True
↪,csvdump=True)
```

---

**create_custom_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new CUSTOM tag.

**Parameters**

- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **csvdump** (bool) – dumps the data in csv
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
Tag ID

**Example**

To create a custom tag 'testing' to user 123

```
>>> self.rs.tags.create_custom_tag('testing','something',123,"#648d9f",True)
```

**Note:** You can also dump the tag id in a csv using the `csvdump=True` argument

```
>>> self.rs.tags.create_custom_tag('testing','something',123,"#648d9f",True,
↪csvdump=True)
```

**create_remediation_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new REMEDIATION tag.

**Parameters**

- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – dumps the data in csv

**Return type**
int

**Returns**
The new tag ID will be returned.

**Example**

To create a remediation tag 'testing' to user 123

```
>>> self.rs.tags.create_remediation_tag('testing','something',123,"#648d9f",
↪True)
```

**Note:** You can also dump the tag id in a csv using the `csvdump=True` argument

```
>>> self.rs.tags.create_remediation_tag('testing','something',123,"#648d9f",
↪True,csvdump=True)
```

**create_people_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new PEOPLE tag.

**Parameters**

- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The new tag ID will be returned.

---

**Example**

To create a people tag 'testing' to user 123

```
>>> self.rs.tags.create_people_tag('testing', 'something', 123, "#648d9f", True)
```

---

**Note:** You can also dump the tag id in a csv using the csvdump=True argument

```
>>> self.rs.tags.create_people_tag('testing', 'something', 123, "#648d9f", True,
↪csvdump=True)
```

---

**create_project_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new PROJECT tag.

**Parameters**

- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – dumps the data in csv

**Return type**
`int`

**Returns**
The new tag ID will be returned.

---

**Example**

To create a project tag 'testing' to user 123

```
>>> self.rs.tags.create_project_tag('testing','something',123,"#648d9f",True)
```

---

**Note:** You can also dump the tag id in a csv using the `csvdump=True` argument

```
>>> self.rs.tags.create_project_tag('testing','something',123,"#648d9f",True
↪csvdump=True)
```

---

**create_scanner_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new SCANNER tag.

**Parameters**

- **name** (`str`) – Name of tag
- **desc** (`str`) – Description of tag
- **owner** (`str`) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (`str`) – Hex value of the color to be used for this tag.
- **locked** (`bool`) – Reflects whether or not the tag should be locked.
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The new tag ID will be returned.

---

**Example**

To create a scanner tag 'testing' to user 123

```
>>> self.rs.tags.create_scanner_tag('testing','something',123,"#648d9f",True)
```

---

**Note:** You can also dump the tag id in a csv using the `csvdump=True` argument

```
>>> self.rs.tags.create_scanner_tag('testing','something',123,"#648d9f",True
↪csvdump=True)
```

---

**create_cmdb_tag**(*name*, *desc*, *owner*, *color='#648d9f'*, *locked=False*, *csvdump=False*, *client_id=None*)

Create a new CMDB tag.

**Parameters**

- **name** (str) – Name of tag
- **desc** (str) – Description of tag
- **owner** (str) – The owner(s) of the tag, represented by user IDs, delimited by commas. Ex: "1234,567,890"
- **color** (str) – Hex value of the color to be used for this tag.
- **locked** (bool) – Reflects whether or not the tag should be locked.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The new tag ID will be returned.

---

**Example**

To create a cmdb tag 'testing' to user 123

```
>>> self.rs.tags.create_cmdb_tag('testing','something',123,"#648d9f",True)
```

---

**Note:** You can also dump the tag id in a csv using the csvdump=True argument

```
>>> self.rs.tags.create_cmdb_tag('testing','something',123,"#648d9f",True,
↪csvdump=True)
```

---

**lock_tag**(*tag_id*, *csvdump=False*, *client_id=None*)

Lock an existing tag.

**Parameters**

- **tag_id** (int) – The tag ID to be locked.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The tag ID

**Example**

To lock a tag id 123

```
>>> self.rs.tags.lock_tag(123)
```

**Note:** You can also dump the tag data in a csv after locking by simply providing `csvdump=True` argument

```
>>> self.rs.tags.lock_tag(123,csvdump=True)
```

**unlock_tag**(*tag_id*, *csvdump=False*, *client_id=None*)

Unlock an existing tag.

**Parameters**

- **tag_id** (int) – The tag ID to be unlocked.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The tag ID

**Example**

To unlock a tag id 123

```
>>> self.rs.tags.unlock_tag(123)
```

**Note:** You can also dump the tag data in a csv after unlocking by simply providing `csvdump=True` argument

```
>>> self.rs.tags.unlock_tag(123,csvdump=True)
```

### 1.2.17 Workflows (risksense_api.__subject.__workflows.__workflows)

**Workflows module defined for different workflows related api endpoints.**

**class** risksense_api.__subject.__workflows.__workflows.**Workflows**(*profile*)

Bases: Subject

**Class for Worflow function defintions**.

To utlise workflow function:

**Parameters**
**profile** (object) – Profile Object

**Usage:**

```
self.{risksenseobjectname}.workflows.{function}
```

---

**Examples**

To get model for workflow using `get_model()` function

```
>>> self.{risksenseobject}.workflows.get_model()
```

---

**class OverrideControl**

Bases: `object`

**NONE = 'NONE'**

**AUTHORIZED = 'AUTHORIZED'**

**class Workflowtype**

Bases: `object`

**FALSEPOSITIVE = 'falsePositive'**

**REMEDIATION = 'remediation'**

**ACCEPTANCE = 'acceptance'**

**SEVERITYCHANGE = 'severityChange'**

**__init__**(*profile*)

Initialization of Workflows object.

**Parameters**
**profile** (`object`) – Profile Object

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns workflows based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (`list`) – A list of dictionaries containing filter parameters.
- **projection** (`str`) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_size** (`int`) – The number of results per page to be returned.
- **sort_field** (`str`) – The field to be used for sorting results returned.
- **sort_dir** (`str`) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **csvdump** (`bool`) – dumps the data in csv
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`list`

**Returns**
A list containing all workflows returned by the search using the filter provided.

---

**Example**

An example to search for workflow data is

```
>>> self.{risksenseobject}.workflow.search([])
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.{risksenseobject}.workflow.search([],csvdump=True)
```

---

`list_workflowbatch_model`(*client_id=None*)

Get available projections and models for Workflowbatch.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Workflow batch models and projections

---

**Example**

An example to use get_model is

```
>>> self.{risksenseobject}.workflowbatch.get_model()
```

---

`getexporttemplate`(*client_id=None*)

Gets configurable export template for workflows.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

---

**Example**

An example to use getexporttemplate

```
>>> self.{risksenseobject}.workflows.getexporttemplate()
```

This gets all the export templates for workflows

---

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*)

Initiates an export job on the platform for workflow(s) based on the provided filter(s), by default fetches all the columns data.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Available options ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform from is returned.

---

**Example**

An example to use export is

```
>>> self.{risksenseobject}.workflows.export([],'testingexport')
```

You can change the filetype to any of the names above or even the other positional arguments as mentioned

```
>>> self.{risksenseobject}.workflows.export([],'testingexport',file_
↪type=ExportFileType.JSON)
```

---

**list_workflowbatch_filter_fields**(*client_id=None*)

List filter endpoints for workflow batch.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON output from the platform is returned, listing the available filters.

---

**Examples**

```
>>>  self.{risksenseobject}.workflows.list_workflowbatch_filter_fields()
```

---

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns workflows based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_num** (int) – The page number of results to be returned.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Example**

An example to get single search page of workflows data

```
>>> self.{risksenseobject}.workflows.get_single_search_page([])
```

You can also try changing the other arguments to your liking to reflect the data as you suffice such as change page_size or page_num etc.

```
>>> self.{risksenseobject}.workflows.get_single_search_page([],page_num=2,page_
↪size=10)
```

---

**suggest**(*search_filter_1*, *search_filter_2*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter_1** (list) – Search Filter 1
- **search_filter_2** (dict) – Search Filter 2
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
Value suggestions

---

**Example**

To use suggest function is

```
>>> self.{risksenseobject}.workflows.suggest([],{})
```

---

**request_acceptance**(*search_filter*, *workflow_name*, *description*, *reason*, *finding_type*, *expiration_date=None*, *override_control='AUTHORIZED'*, *compensating_controls='NONE'*, *attachment=None*, *client_id=None*)

Request acceptance for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **search_filter** (list) – A list of dictionaries containing filter parameters.
- **workflow_name** (str) – Workflow Name
- **description** (str) – A description of the request.
- **reason** (str) – A reason for the request.
- **finding_type** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **expiration_date** (typing.Optional[str]) – An expiration date. Should be in "YYYY-MM-DD" format.
- **override_control** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **compensating_controls** (str) – A description of compensating controls applied to this finding. Option available : ("DLP", "Deemed not exploitable", "Endpoint Security", "IDS/IPS", "MFA Enforced", "Multiple: See Description", "Network Firewall", "Network Segmentation", "Other: See Description", "Web Application Firewall" or "NONE")
- **attachment** (typing.Optional[str]) – A path to a file to be uploaded and attached to the request.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To create request acceptance for findings for hostfindings

```
>>> self.rs.workflows.request_acceptance([],'testingforsomething','something',
↪'none',"hostFinding","2022-08-11")
```

---

**request_false_positive**(*finding_type*, *search_filter*, *workflow_name*, *description*, *reason*, *override_control='AUTHORIZED'*, *expiration_date=None*, *attachment=None*, *client_id=None*)

Request false positive for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **finding_type** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **search_filter** (list) – A list of dictionaries containing filter parameters.
- **workflow_name** (str) – Workflow Name

- **description** (str) – A description of the request.
- **reason** (str) – A reason for the request.
- **override_control** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **expiration_date** (typing.Optional[str]) – An expiration date. Should be in "YYYY-MM-DD" format.
- **attachment** (typing.Optional[str]) – A path to a file to be uploaded and attached to the request.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To create request false positive for findings for hostfindings

```
>>> self.rs.workflows.request_false_positive("hostFinding",[],
↪'testingforsomething','something','none',"2022-08-11")
```

---

**request_remediation**(*finding_type*, *search_filter*, *workflow_name*, *description*, *reason*, *override_control='AUTHORIZED'*, *expiration_date=None*, *attachment=None*, *client_id=None*)

Request remediation for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **finding_type** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **search_filter** (list) – A list of dictionaries containing filter parameters.
- **workflow_name** (str) – Workflow Name
- **description** (str) – A description of the request.
- **reason** (str) – A reason for the request.
- **override_control** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **expiration_date** (typing.Optional[str]) – An expiration date. Should be in "YYYY-MM-DD" format.
- **attachment** (typing.Optional[str]) – A path to a file to be uploaded and attached to the request.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To create request remediation for findings for hostfindings

```
>>> self.rs.workflows.request_remediation("hostFinding",[],'testingforsomething
↪','something','none',"2022-08-11")
```

---

**request_severity_change**(*finding_type*, *search_filter*, *workflow_name*, *description*, *reason*, *severity_change*, *override_control='AUTHORIZED'*, *expiration_date=None*, *attachment=None*, *client_id=None*)

Request severity change for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **finding_type** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **search_filter** (list) – A list of dictionaries containing filter parameters.
- **workflow_name** (str) – Workflow Name
- **description** (str) – A description of the request.
- **reason** (str) – A reason for the request.
- **severity_change** (str) – Severity change value.
- **override_control** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **compensating_controls** – Severity change for this finding. Option available : ("1" to "10")
- **expiration_date** (typing.Optional[str]) – An expiration date. Should be in "YYYY-MM-DD" format.
- **attachment** (typing.Optional[str]) – A path to a file to be uploaded and attached to the request.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To create request severity change for findings for hostfindings

```
>>> self.rs.workflows.request_false_positive("hostFinding",[],
↪'testingforsomething','something','4','none',"2022-08-11")
```

---

**reject_workflow**(*filter_request*, *workflowtype*, *description*, *csvdump=False*, *client_id=None*)

Reject a workflow request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rejection.
- **workflowtype** (str) – Type of workflow
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To perform a reject acceptance request for a workflow filter RA#0000028

```
>>> self.rs.workflows.reject_workflow([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}],'acceptance','needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.reject_workflow([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}],'acceptance','needed to test',csvdump=True)
```

---

**rework_workflow**(*filter_request*, *workflow_type*, *description*, *csvdump=False*, *client_id=None*)

Request a rework of a workflow.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rework.
- **csvdump** (bool) – dumps the data in csv
- **workflow_type** (str) – Type of workflow
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

**Example**

To rework an acceptance request RA#0000027

```
>>> self.rs.workflows.rework_workflow([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000027"}],'acceptance','needed to test')
```

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.rework_workflow([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000027"}],'acceptance','needed to test',csvdump=True)
```

**approve_workflow**(*filter_request*, *workflowtype*, *override_exp_date=False*, *expiration_date=datetime.date(2023, 2, 22)*, *csvdump=False*, *client_id=None*, ***kwargs*)

Approve a workflow request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowtype** (str) – Type of workflow
- **override_exp_date** (bool) – True/False indicating whether or not an expiration date should be overridden.
- **expiration_date** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

**Example**

To approve an acceptance request RA#0000028

```
>>> self.rs.workflows.approve_acceptance([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}],'acceptance')
```

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.approve_acceptance([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}],'acceptance',csvdump=True)
```

---

**update_workflow**(*workflowBatchUuid*, *workflowtype*, *name*, *expirationDate*, *description*, *reason*, *compensatingControl*, *overrideControl='AUTHORIZED'*, *csvdump=False*, *client_id=None*, ***kwargs*)

Update a workflow.

**Parameters**

- **workflowBatchUuid** (str) – Workflow UUID
- **workflowtype** (str) – Type of workflow
- **name** (str) – Workflow name
- **expirationDate** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **description** (str) – A description of the rejection.
- **reason** (str) – A reason for the rejection.
- **compensatingControl** (str) – A description of compensating controls applied to this finding. Option available : ("DLP", "Deemed not exploitable", "Endpoint Security", "IDS/IPS", "MFA Enforced", "Multiple: See Description", "Network Firewall", "Network Segmentation", "Other: See Description", "Web Application Firewall" or "NONE")
- **overrideControl** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Keyword Arguments**
**severity** (severity) – The severity number

**Return type**
dict

**Returns**
The jsonified response.

---

**Example**

To Update a false positive workflow of uuid 11ed0429-4937-f454-b7ab-06933745a4d6

```
>>> self.rs.workflows.update_workflow("11ed0429-4937-f454-b7ab-06933745a4d6",
↪"testin","2022-07-31","testingforupdate",'falsePositive',"testing","NONE")
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.rs.workflows.update_workflow("11ed0429-4937-f454-b7ab-06933745a4d6",
↪"testin","2022-07-31","testingforupdate",'falsePositive',"testing","NONE",
↪csvdump=True)
```

---

**map_findings**(*subject*, *filter_request*, *workflowuuid*, *workflowtype*, *client_id=None*)

Map findings to a workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – Workflow type
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To map acceptance workflow to findings

```
>>> self.rs.workflows.map_findings_acceptance("hostFinding",[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"235141285"}],'acceptance',
↪"11ed123b-9d8f-a2f1-b7ab-06933745a4d6")
```

---

**unmap_findings**(*subject*, *filter_request*, *workflowuuid*, *workflowtype*, *client_id=None*)

Unmap findings to a workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – Workflow type
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To unmap findings to acceptance workflow

```
>>> self.rs.workflows.unmap_findings('hostFinding',[{"field":"id","exclusive
↪" False,"operator":"IN","value":"232927123,178257910"}],'11ed0429-4937-f454-
↪b7ab-06933745a4d6','acceptance')
```

---

**get_attachments**(*workflowbatchuuid*, *workflowtype*, *subject*, *client_id=None*)

Get attachments from an acceptance workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflowbatch uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **workflowtype** (str) – Type of workflow
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

---

**Example**

To use attachment function

```
>>> self.rs.workflows.get_attachments_acceptance('11ed0429-4937-f454-b7ab-
↪06933745a4d6','hostFinding')
```

---

**download_workflowbatch_attachments**(*fileuuid*, *subject*, *workflowtype*, *workflowcategory='CLOSE_REQUEST'*, *client_id=None*)

Download attachments from a workflow

**Parameters**

- **fileuuid** (str) – File uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **workflowtype** (str) – Type of workflow
- **workflowcategory** (str) – Workflow category either CLOSE_REQUEST or CHANGE_REQUEST
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success

---

**Example**

To download workflowbatch attachments for hostfindings

```
>>> self.rs.workflows.download_workflowbatch_attachments_acceptance('test123',
↪'hostFinding','acceptance)
```

---

**attach_files**(*workflowbatchuuid*, *subject*, *workflowtype*, *file_name*, *path_to_file*, *client_id=None*)

Attach a file to workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **workflowtype** (str) – Type of workflow
- **file_name** (str) – The name to be used for the uploaded file.
- **path_to_file** (str) – Full path to the file to be uploaded.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether attachment is done

---

**Example**

To attach file to workflow

```
>>> self.rs.workflows.attach_files('11ed042d-1fcc-fdfe-b7ab-06933745a4d6',
↪'hostFinding','acceptance','test.csv','test.csv')
```

---

**detach_files**(*workflowbatchuuid*, *attachmentuuids*, *workflowtype*, *subject*, *client_id=None*)

Detach a file from acceptance workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **attachmentuuids** (list) – Attchment UUID
- **client_id** – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether detachment is done

---

**Example**

**To detach files from workflow**
self.rs.workflows.detach_files_acceptance('11ed042d-1fcc-fdfe-b7ab-06933745a4d6',["bfe66d56-b7da-4577-a86f-da5283505ea7"],'hostFinding')

---

**Parameters**
**workflowtype** (str) –

**reject_acceptance**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Reject an acceptance request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rejection.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To perform a reject acceptance request for a workflow filter RA#0000028

```
>>> self.rs.workflows.reject_acceptance([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}],'needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.reject_acceptance([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}],'needed to test',csvdump=True)
```

---

**reject_false_positive**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Reject a false positive request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rejection.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To perform a reject false positive request for a workflow FP#0000020

```
>>> self.rs.workflows.reject_false_positive([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"FP#0000020"}],'needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.reject_false_positive([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"FP#0000020"}],'needed to test',csvdump=True)
```

---

**reject_remediation**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Reject a remediation request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rejection.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To perform a reject remediation request for a workflow RM#0000044

```
>>> self.rs.workflows.reject_remediation([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RM#0000044"}],'needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.reject_remediation([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RM#0000044"}],'needed to test',csvdump=True)
```

---

**reject_severity_change**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Reject a severity change request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rejection.

- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To perform a reject severity change request for a workflow SC#0000061

```
>>> self.rs.workflows.reject_severity_change([{"field":"generated_id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"SC#0000061"}],'needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.reject_severity_change([{"field":"generated_id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"SC#0000061"}],'needed to test',csvdump=True)
```

---

**rework_acceptance**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Request a rework of an acceptance.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rework.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To rework an acceptance request RA#0000027

```
>>> self.rs.workflows.rework_acceptance([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000027"}],'needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.rework_acceptance([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000027"}],'needed to test',csvdump=True)
```

---

**rework_false_positive**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Request a rework of a false positive.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rework.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To rework an false positive request FP#0000019

```
>>> self.rs.workflows.rework_false_positive([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"FP#0000019"}],'needed to test')
```

---

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.rework_false_positive([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"FP#0000019"}],'needed to test',csvdump=True)
```

---

**rework_remediation**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Request a rework of a remediation.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rework.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To rework an remediation request RM#0000043

```
>>> self.rs.workflows.rework_remediation([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RM#0000043"}],'needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.rework_remediation([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RM#0000043"}],'needed to test',csvdump=True)
```

---

**rework_severity_change**(*filter_request*, *description*, *csvdump=False*, *client_id=None*)

Request a rework of a severity change.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **description** (str) – A description of the rework.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To rework a severity change request SC#0000027

```
>>> self.rs.workflows.rework_severity_change([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"SC#0000027"}],'needed to test')
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.rework_severity_change([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"SC#0000027"}],'needed to test',csvdump=True)
```

---

**approve_acceptance**(*filter_request*, *override_exp_date=False*, *expiration_date=datetime.date(2023, 2, 22)*, *csvdump=False*, *client_id=None*)

Approve a acceptance request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **override_exp_date** (bool) – True/False indicating whether or not an expiration date should be overridden.
- **expiration_date** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To approve an acceptance request RA#0000028

```
>>> self.rs.workflows.approve_acceptance([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}])
```

---

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.approve_acceptance([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RA#0000028"}],csvdump=True)
```

---

**approve_false_positive**(*filter_request*, *override_exp_date=False*, *expiration_date=datetime.date(2023, 2, 22)*, *csvdump=False*, *client_id=None*)

Approve a false positive change request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **override_exp_date** (bool) – True/False indicating whether or not an expiration date should be overridden.
- **expiration_date** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To approve a false positive request FP#0000020

```
>>> self.rs.workflows.approve_false_positive([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"FP#0000020"}])
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.approve_false_positive([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"FP#0000020"}],csvdump=True)
```

---

**approve_remediation**(*filter_request*, *override_exp_date=False*, *expiration_date=datetime.date(2023, 2, 22)*, *csvdump=False*, *client_id=None*)

Approve a remediation request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **override_exp_date** (bool) – True/False indicating whether or not an expiration date should be overridden.
- **expiration_date** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To approve an remediation request RM#0000044

```
>>> self.rs.workflows.approve_remediation([{"field":"generated_id","exclusive
↪" False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RM#0000044"}])
```

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.approve_remediation([{"field":"generated_id","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"RM#0000044"}],csvdump=True)
```

---

**approve_severity_change**(*filter_request*, *override_exp_date=False*, *expiration_date=datetime.date(2023, 2, 22)*, *csvdump=False*, *client_id=None*)

Approve a severity change request.

**Parameters**

- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **override_exp_date** (bool) – True/False indicating whether or not an expiration date should be overridden.
- **expiration_date** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID from the platform is returned.

---

**Example**

To approve an severity change request SC#0000061

```
>>> self.rs.workflows.approve_severity_change([{"field":"generated_id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"SC#0000061"}])
```

---

---

**Note:** You can also dump the job id in a csv using:obj:*csvdump=True* argument

```
>>> self.rs.workflows.approve_severity_change([{"field":"generated_id",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"SC#0000061"}],csvdump=True)
```

---

**update_acceptance_workflow**(*workflowBatchUuid*, *name*, *expirationDate*, *description*, *reason*, *compensatingControl*, *overrideControl='AUTHORIZED'*, *csvdump=False*, *client_id=None*)

Update an acceptance workflow.

**Parameters**

- **workflowBatchUuid** (str) – Workflow UUID
- **name** (str) – Workflow name

- **expirationDate** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **description** (str) – A description of the rejection.
- **reason** (str) – A reason for the rejection.
- **compensatingControl** (str) – A description of compensating controls applied to this finding. Option available : ("DLP", "Deemed not exploitable", "Endpoint Security", "IDS/IPS", "MFA Enforced", "Multiple: See Description", "Network Firewall", "Network Segmentation", "Other: See Description", "Web Application Firewall" or "NONE")
- **overrideControl** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The jsonified response.

---

**Example**

To update an acceptance workflow of uuid 11ed0429-4937-f454-b7ab-06933745a4d6

```
>>> self.rs.workflows.update_acceptance_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","2022-07-31","testingforupdate","testing","NONE")
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.rs.workflows.update_acceptance_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","2022-07-31","testingforupdate","testing","NONE",
↪csvdump=True)
```

---

**update_falsepositive_workflow**(*workflowBatchUuid*, *name*, *expirationDate*, *description*, *reason*, *compensatingControl*, *overrideControl='AUTHORIZED'*, *csvdump=False*, *client_id=None*)

Update a false positive workflow.

**Parameters**

- **workflowBatchUuid** (str) – Workflow UUID
- **name** (str) – Workflow name
- **expirationDate** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **description** (str) – A description of the rejection.
- **reason** (str) – A reason for the rejection.

- **compensatingControl** (str) – A description of compensating controls applied to this finding. Option available : ("DLP", "Deemed not exploitable", "Endpoint Security", "IDS/IPS", "MFA Enforced", "Multiple: See Description", "Network Firewall", "Network Segmentation", "Other: See Description", "Web Application Firewall" or "NONE")
- **overrideControl** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The jsonified response.

---

**Example**

To Update a false positive workflow of uuid 11ed0429-4937-f454-b7ab-06933745a4d6

```
>>> self.rs.workflows.update_falsepositive_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","2022-07-31","testingforupdate","testing","NONE")
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.rs.workflows.update_falsepositive_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","2022-07-31","testingforupdate","testing","NONE",
↪csvdump=True)
```

---

**update_remediation_workflow**(*workflowBatchUuid*, *name*, *expirationDate*, *description*, *reason*, *compensatingControl*, *overrideControl='AUTHORIZED'*, *csvdump=False*, *client_id=None*)

Update a remediation workflow.

**Parameters**

- **workflowBatchUuid** (str) – Workflow UUID
- **name** (str) – Workflow name
- **expirationDate** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **description** (str) – A description of the rejection.
- **reason** (str) – A reason for the rejection.
- **compensatingControl** (str) – A description of compensating controls applied to this finding. Option available : ("DLP", "Deemed not exploitable", "Endpoint Security", "IDS/IPS", "MFA Enforced", "Multiple: See Description", "Network Firewall", "Network Segmentation", "Other: See Description", "Web Application Firewall" or "NONE")
- **overrideControl** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')

- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The jsonified response.

---

**Example**

To update an remediation workflow of uuid 11ed0429-4937-f454-b7ab-06933745a4d6

```
>>> self.rs.workflows.update_remediation_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","2022-07-31","testingforupdate","testing","NONE")
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.rs.workflows.update_remediation_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","2022-07-31","testingforupdate","testing","NONE",
↪csvdump=True)
```

---

**update_severitychange_workflow**(*workflowBatchUuid*, *name*, *expirationDate*, *description*, *reason*, *severity*, *compensatingControl*, *overrideControl='AUTHORIZED'*, *csvdump=False*, *client_id=None*)

Update an severity change workflow.

**Parameters**

- **workflowBatchUuid** (str) – Workflow UUID
- **name** (str) – Workflow name
- **expirationDate** (str) – An expiration date for the approval. Should be in "YYYY-MM-DD" format.
- **description** (str) – A description of the rejection.
- **reason** (str) – A reason for the rejection.
- **severity** (str) – Severity change value
- **compensatingControl** (str) – A description of compensating controls applied to this finding. Option available : ("DLP", "Deemed not exploitable", "Endpoint Security", "IDS/IPS", "MFA Enforced", "Multiple: See Description", "Network Firewall", "Network Segmentation", "Other: See Description", "Web Application Firewall" or "NONE")
- **overrideControl** (str) – A description of override controls applied to this finding. Option available : ('NONE', 'AUTHORIZED')
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The jsonified response.

---

**Example**

To update a severity change workflow of uuid 11ed0429-4937-f454-b7ab-06933745a4d6

```
>>> self.rs.workflows.update_severitychange_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","5","2022-07-31","testingforupdate","testing","NONE")
```

---

**Note:** You can also dump the search based data in a csv by simply providing `csvdump=True` argument

```
>>> self.rs.workflows.update_severitychange_workflow("11ed0429-4937-f454-b7ab-
↪06933745a4d6","testin","5","2022-07-31","testingforupdate","testing","NONE",
↪csvdump=True)
```

---

**map_findings_acceptance**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='acceptance'*, *client_id=None*)

Map findings to an acceptance workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default acceptance
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To map acceptance workflow to findings

```
>>> self.rs.workflows.map_findings_acceptance("hostFinding",[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"235141285"}],"11ed123b-9d8f-a2f1-
↪b7ab-06933745a4d6")
```

---

**map_findings_severitychange**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='severityChange'*, *client_id=None*)

Map findings to an severity change workflow for applicationFindings / hostfFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default severity change
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To map severity change workflow to findings

```
>>> self.rs.workflows.map_findings_severitychange("hostFinding",[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"235141285"}],"11ed123b-9d8f-a2f1-
↪b7ab-06933745a4d6")
```

---

**map_findings_falsepositive**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='falsePositive'*, *client_id=None*)

Map findings to an false positive workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default false positive
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To map false positive workflow to findings

```
>>> self.rs.workflows.map_findings_falsepositive("hostFinding",[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"235141285"}],"11ed123b-9d8f-a2f1-
↪b7ab-06933745a4d6")
```

---

**map_findings_remediation**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='remediation'*, *client_id=None*)

Map findings to an remediation workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default remediation
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To map remediation workflow to findings

```
>>> self.rs.workflows.map_findings_remediation("hostFinding",[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"235141285"}],"11ed123b-9d8f-a2f1-
↪b7ab-06933745a4d6")
```

---

**unmap_findings_acceptance**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='acceptance'*, *client_id=None*)

Unmap findings to an acceptance workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default acceptance
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To unmap findings to workflow

```
>>> self.rs.workflows.unmap_findings_acceptance('hostFinding',[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"232927123,178257910"}],'11ed0429-
↪4937-f454-b7ab-06933745a4d6')
```

---

**unmap_findings_severitychange**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='severityChange'*, *client_id=None*)

Unmap findings to a severity change workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default acceptance
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

---

**Example**

To unmap findings to workflow

```
>>> self.rs.workflows.unmap_findings_severitychange('hostFinding',[{"field":"id
↪","exclusive":False,"operator":"IN","value":"232927123,178257910"}],'11ed0429-
↪4937-f454-b7ab-06933745a4d6')
```

---

**unmap_findings_falsepositive**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='falsePositive'*, *client_id=None*)

Unmap findings to a false positive workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default false positive
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

**Example**

To unmap findings to workflow

```
>>> self.rs.workflows.unmap_findings_falsepositive('hostFinding',[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"232927123,178257910"}],'11ed0429-
↪4937-f454-b7ab-06933745a4d6')
```

**unmap_findings_remediation**(*subject*, *filter_request*, *workflowuuid*, *workflowtype='remediation'*, *client_id=None*)

Unmap findings to a remediation workflow for applicationFindings / hostFindings as defined in the filter_request parameter.

**Parameters**

- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **filter_request** (list) – A list of dictionaries containing filter parameters.
- **workflowuuid** (str) – Workflow Name
- **workflowtype** (str) – By default remediation
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether request occured

**Example**

To unmap findings to workflow

```
>>> self.rs.workflows.unmap_findings_remediation('hostFinding',[{"field":"id",
↪"exclusive":False,"operator":"IN","value":"232927123,178257910"}],'11ed0429-
↪4937-f454-b7ab-06933745a4d6')
```

**get_attachments_acceptance**(*workflowbatchuuid*, *subject*, *client_id=None*)

Get attachments from an acceptance workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflowbatch uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Example**

To use attachment function

```
>>> self.rs.workflows.get_attachments_acceptance('11ed0429-4937-f454-b7ab-
↪06933745a4d6', 'hostFinding')
```

---

**get_attachments_severitychange**(*workflowbatchuuid*, *subject*, *client_id=None*)

Get attachments from a severity change workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflowbatch uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

---

**Example**

To use attachment function

```
>>> self.rs.workflows.get_attachments_severitychange('11ed0429-4937-f454-b7ab-
↪06933745a4d6', 'hostFinding')
```

---

**get_attachments_remediation**(*workflowbatchuuid*, *subject*, *client_id=None*)

Get attachments from a remediation workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflowbatch uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

---

**Example**

```
>>> self.rs.workflows.get_attachments_remediation('11ed0429-4937-f454-b7ab-
↪06933745a4d6', 'hostFinding')
```

---

**get_attachments_falsepositive**(*workflowbatchuuid*, *subject*, *client_id=None*)

Get attachments from a false positive workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflowbatch uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

---

**Example**

To use attachment function

```
>>> self.rs.workflows.get_attachments_falsepositive('11ed0429-4937-f454-b7ab-
↪06933745a4d6','hostFinding')
```

---

**download_workflowbatch_attachments_acceptance**(*fileuuid*, *subject*, *workflowcategory='CLOSE_REQUEST'*, *client_id=None*)

Download attachments from an acceptance workflow

**Parameters**

- **fileuuid** (str) – File uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **workflowcategory** (str) – Workflow category either CLOSE_REQUEST or CHANGE_REQUEST
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success

---

**Example**

To download workflowbatch attachments for hostfindings

```
>>> self.rs.workflows.download_workflowbatch_attachments_acceptance('test123',
↪'hostFinding')
```

---

**download_workflowbatch_attachments_severitychange**(*fileuuid*, *subject*, *workflowcategory='CLOSE_REQUEST'*, *client_id=None*)

Download attachments from an severity change workflow

**Parameters**

- **fileuuid** (str) – File uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **workflowcategory** (str) – Workflow category either CLOSE_REQUEST or CHANGE_REQUEST
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success

---

**Example**

To download workflowbatch attachments for hostfindings

```
>>> self.rs.workflows.download_workflowbatch_attachments_severitychange('test123
↪' 'hostFinding')
```

---

**download_workflowbatch_attachments_falsepositive**(*fileuuid*, *subject*, *workflowcategory='CLOSE_REQUEST'*, *client_id=None*)

Download attachments from an False positive workflow

**Parameters**

- **fileuuid** (str) – File uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **workflowcategory** (str) – Workflow category either CLOSE_REQUEST or CHANGE_REQUEST
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success

---

**Example**

To download workflowbatch attachments for hostfindings

```
>>> self.rs.workflows.download_workflowbatch_attachments_falsepositive('test123
↪' 'hostFinding')
```

---

**download_workflowbatch_attachments_remediation**(*fileuuid*, *subject*, *workflowcategory='CLOSE_REQUEST'*, *client_id=None*)

Download attachments from an remediation workflow

**Parameters**

- **fileuuid** (str) – File uuid
- **subject** (str) – Subject whether hostFinding or applicationFinding
- **workflowcategory** (str) – Workflow category either CLOSE_REQUEST or CHANGE_REQUEST
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success

**Example**

To download workflowbatch attachments for hostfindings

```
>>> self.rs.workflows.download_workflowbatch_attachments_remediation('test123',
↪'hostFinding')
```

**attach_files_acceptance**(*workflowbatchuuid*, *subject*, *file_name*, *path_to_file*, *client_id=None*)

Attach an acceptance file to workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **file_name** (str) – The name to be used for the uploaded file.
- **path_to_file** (str) – Full path to the file to be uploaded.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether attachment is done

**Example**

To attach file to workflow

```
>>> self.rs.workflows.attach_files_acceptance('11ed042d-1fcc-fdfe-b7ab-
↪06933745a4d6','hostFinding','test.csv','test.csv')
```

**attach_files_remediation**(*workflowbatchuuid*, *subject*, *file_name*, *path_to_file*, *client_id=None*)

Attach a file to remediation workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **file_name** (str) – The name to be used for the uploaded file.
- **path_to_file** (str) – Full path to the file to be uploaded.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether attachment is done

**Example**

To attach file to workflow

```
>>> self.rs.workflows.attach_files_remediation('11ed042d-1fcc-fdfe-b7ab-
↪06933745a4d6','hostFinding','test.csv','test.csv')
```

**attach_files_falsepositive**(*workflowbatchuuid*, *subject*, *file_name*, *path_to_file*, *client_id=None*)

Attach a file to false positive workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **file_name** (str) – The name to be used for the uploaded file.
- **path_to_file** (str) – Full path to the file to be uploaded.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether attachment is done

**Example**

To attach file to workflow

```
>>> self.rs.workflows.attach_files_falsepositive('11ed042d-1fcc-fdfe-b7ab-
↪06933745a4d6','hostFinding','test.csv','test.csv')
```

**attach_files_severitychange**(*workflowbatchuuid*, *subject*, *file_name*, *path_to_file*, *client_id=None*)

Attach a file to severiy change workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **file_name** (str) – The name to be used for the uploaded file.
- **path_to_file** (str) – Full path to the file to be uploaded.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether attachment is done=

**Example**

To attach file to workflow

```
>>> self.rs.workflows.attach_files_severitychange('11ed042d-1fcc-fdfe-b7ab-
↪06933745a4d6','hostFinding','test.csv','test.csv')
```

**detach_files_acceptance**(*workflowbatchuuid*, *attachmentuuids*, *subject*, *client_id=None*)

Detach a file from acceptance workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **attachmentuuids** (list) – Attchment UUID
- **client_id** – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether detachment is done

**Example**

To detach files from workflow

```
>>> self.rs.workflows.detach_files_acceptance('11ed042d-1fcc-fdfe-b7ab-
↪06933745a4d6',["bfe66d56-b7da-4577-a86f-da5283505ea7"],'hostFinding')
```

**detach_files_falsepositive**(*workflowbatchuuid*, *attachmentuuids*, *subject*, *client_id=None*)

Detach files from false positive workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **attachmentuuids** (list) – Attchment UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether detachment is done

**Example**

To detach files from workflow

```
>>> self.rs.workflows.detach_files_falsepositive( "11ed042d-006c-358a-b7ab-
↪06933745a4d6",[  "a59fd436-b2ba-46f9-9962-ea227426532d"],'hostFinding')
```

---

**detach_files_remediation**(*workflowbatchuuid*, *attachmentuuids*, *subject*, *client_id=None*)

Detach files from remediation workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **attachmentuuids** (list) – Attchment UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether detachment is done

---

**Example**

To detach files from workflow

```
>>> self.rs.workflows.detach_files_remediation( "11ed042d-1fcc-fdfe-b7ab-
↪06933745a4d6",[  "95e082c9-b4f6-46ce-baab-538375db85d2"],'hostFinding')
```

---

**detach_files_severitychange**(*workflowbatchuuid*, *attachmentuuids*, *subject*, *client_id=None*)

Detach files from severity change workflow

**Parameters**

- **workflowbatchuuid** (str) – Workflow UUID
- **subject** (str) – Finding type. Possible options : ("hostFinding" or "applicationFinding")
- **attachmentuuids** (list) – Attchment UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Success whether detachment is done

---

**Example**

To detach files from workflow

```
>>> self.rs.workflows.detach_files_severitychange('11ed042d-15bb-90f6-b7ab-
↪06933745a4d6',[ "c4e992e7-f911-442e-92cf-b8db003eafdb"],'hostFinding')
```

---

### 1.2.18 Assessments (risksense_api.__subject.__assessments.__assessments)

Assessment module defined for different assessment related api endpoints.

**class** risksense_api.__subject.__assessments.__assessments.**Assessments**(*profile*)

Bases: Subject

Class for assessment function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise assessment function:

**Usage:**
self.{risksenseobjectname}.assessments.{function}

---

**Examples**

To create an assessment using *create()* function

```
>>> self.{risksenseobjectname}.assessments.create(args)
```

---

**__init__**(*profile*)

Initialization of Assessments object.

**Parameters**
**profile** (object) – Profile Object

**create**(*name*, *start_date*, *notes=''*, *client_id=None*, *csvdump=False*)

Creates an assessment.

**Parameters**

- **name** (str) – The name for new assessment.
- **start_date** (datetime.date) – The date for the new assessment. Should be in "YYYY-MM-DD" format.
- **notes** (str) – Any notes to associated with the assessment.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
Assessment job id

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.create('hello','2022-02-11',
↪'testingtherisksense')
```

---

**Note:** You can also dump the data of the assessment job id in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.assessments.create('hello','2022-02-11',
↪'testingtherisksense',csvdump=True)
```

**update**(*assessment_id*, *client_id=None*, *csvdump=False*, ***kwargs*)

Update an assessment

**Parameters**

- **assessment_id** (int) – The assessment ID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Keyword Arguments**

- **name** (*str*) – The name to assign to the assessment.
- **start_date** (*str*) – The start date to assign to the assessment. Should be in "YYYY-MM-DD" format.
- **notes** (*str*) – Notes to assign to the assessment.

**Return type**
int

**Returns**
The job ID is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.update(216917, start_date='2022-
↪08-01', name='testcase_aug_test', notes='')
```

**Note:** You can also dump the data of the assessment job id in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.assessments.update(216917, start_date='2022-08-01',␣
↪name='testcase_aug_test',csvdump=True)
```

**delete**(*assessment_id*, *client_id=None*, *csvdump=False*)

Deletes an assessment.

**Parameters**

- **assessment_id** (int) – Assessment ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
The job ID is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.delete(216917)
```

**Note:** You can also dump the data of the assessment that will be deleted in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.assessments.delete(216917,csvdump=True)
```

`get_model`(*client_id=None*)

Get available projections and models for Assessments.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Assessment projections and models are returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.get_model(123)
```

`list_assessment_filter_fields`(*client_id=None*)

List filter endpoints.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON output from the platform is returned, listing the available filters.

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.list_assessment_filter_
↪fields(123)
```

`suggest`(*search_filter*, *suggest_filter*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter** (list) – Search Filter
- **suggest_filter** (dict) – Suggest Filter

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Value suggestions

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.suggest([],{"field":"name",
↪"exclusive":False,"operator":"WILDCARD","value":"testcase_aug_test,",
↪"implicitFilters":[]})
```

**get_single_search_page**(*search_filters*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns assessments based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_num** (int) – Page number of results to be returned.
- **page_size** (int) – Number of results to be returned per page.
- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The paginated JSON response from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.get_single_search_page([{"field
↪":"name","exclusive":False,"operator":"WILDCARD","value":"testcase_aug_test,",
↪"implicitFilters":[]}])
```

**search**(*search_filters*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*, *csvdump=False*)

Searches for and returns assessments based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – Name of field to sort results on.

- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
list

**Returns**
A list containing all hosts returned by the search using the filter provided.

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.search([{"field":"name",
↪"exclusive":False,"operator":"WILDCARD","value":"testcase_aug_test,",
↪"implicitFilters":[]}])
```

**Note:** You can also dump the data of the assessment searched in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.assessments.search([{"field":"name","exclusive
↪":False,"operator":"WILDCARD","value":"testcase_aug_test,","implicitFilters
↪":[]}],csvdump=True)
```

**update_assessment_status**(*assessment_id*, *status*, *client_id=None*, *csvdump=False*)

Update the status of the assessment

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **status** (str) – Update the status either "LOCKED" or "UNLOCKED"
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
The jsonified data of the status

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.update_assessment_status(216917)
```

**Note:** You can also dump the data of the assessment that will be updated in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.assessments.update_assessment_status(216917)
```

---

`get_assessment_history`(*assessment_id*, *csvdump=False*, *client_id=None*)

Get history of assessments

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **csvdump** (bool) – Whether to dump the assessment history in a csv, true to dump and false to not dump
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The jsonified data of the status

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.get_assessment_history(216917)
```

---

**Note:** You can also dump the data of the assessment history in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.assessments.get_assessment_history(216917,
↪csvdump=True)
```

---

`list_attachments`(*assessment_id*, *csvdump=False*, *client_id=None*)

Lists attachments associated with an assessment.

**Parameters**

- **assessment_id** (int) – The assessment ID
- **csvdump** (bool) – Whether to dump the attachment data in a csv, true to dump and false to not dump
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list of attachments associated with the assessment is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.list_attachments(216917)
```

---

**Note:** You can also dump the data of the assessment attachment in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.assessments.list_attachments(216917)
```

---

**get_attachment**(*assessment_id*, *attachment_uuid*, *filename*, *client_id=None*)

Download an attachment associated with an assessment.

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **attachment_uuid** (str) – The unique ID associated with the attachment.
- **filename** (str) – The filename to be used for the downloaded file along with the file type extension.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
True/False indicating whether or not the operation was successful.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.get_attachment(216917, '123-456')
```

---

**get_attachment_metadata**(*assessment_id*, *attachment_uuid*, *client_id=None*)

Get the metadata associated with an assessment's attachment.

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **attachment_uuid** (str) – The unique ID associated with the attachment.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
A dictionary containing the metadata for the attachment is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.get_attachment_metadata(216917,
↪'123-456')
```

---

**edit_network_internalreport**(*assessment_id*, *conclusion*, *constraints*, *client_id=None*)

Edit network internal report for an assessment

**Parameters**

- **assessment_id** (int) – The assessment ID.

- **conclusion** (str) – Conclusion statement for internal report
- **constraints** (str) – Constraints statement for internal report
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified data of the response

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.edit_network_
↪internalreport(216917, 'abc', 'abc')
```

**edit_application_report**(*assessment_id*, *conclusion*, *constraints*, *client_id=None*)

Edit application report for an assessment

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **conclusion** (str) – Conclusion statement for internal report
- **constraints** (str) – Constraints statement for internal report
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified data of the response

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.edit_application_report(216917,
↪'abc', 'abc')
```

**edit_networkexternal_report**(*assessment_id*, *conclusion*, *constraints*, *client_id=None*)

Edit network external report for an assessment

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **conclusion** (str) – Conclusion statement for internal report
- **constraints** (str) – Constraints statement for internal report
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified data of the response

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.edit_networkexternal_
↪report(216917,'abc','abc')
```

`lock_assessment`(*assessment_id*, *client_id=None*)

Lock an assessment

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The jsonified data of the status

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.lock_assessment(216917)
```

`unlock_assessment`(*assessment_id*, *client_id=None*)

Unlock an assessment

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The jsonified data of the status

**Examples**

```
>>> apiobj = self.{risksenseobject}.assessments.unlock_assessment(216917)
```

### 1.2.19 Networks (risksense_api.__subject.__networks.__networks)

Network module defined for different network related api endpoints.

**class** risksense_api.__subject.__networks.__networks.**Networks**(*profile*)

Bases: Subject

Class for network function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise network function:

**Usage:**
self.{risksenseobjectname}.networks.{function}

---

**Examples**

To create a network using *create()* function

```
>>> self.{risksenseobjectname}.networks.create(args)
```

---

**__init__**(*profile*)

Initialization of Networks object.

**Parameters**
**profile** (object) – Profile Object

**list_network_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON output from the platform is returned, listing the available filters.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.networks.list_network_filter_fields(client_
↪id=124)
```

---

**create**(*name*, *client_id=None*, *csvdump=False*)

Create a new network.

**Parameters**

- **name** (str) – The name for the new network.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
The new network job ID.

**Examples**

```
>>> apiobj = self.{risksenseobject}.networks.create('test_aug', 'IP')
```

**Note:** You can also dump the data of the network job id in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.networks.create('test_aug', 'IP',csvdump=True)
```

**update**(*network_id*, *name*, *client_id=None*, *csvdump=False*)

Update an existing network.

**Parameters**

- **network_id** (int) – The network ID.
- **name** (str) – A new name for the network.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
Network job id

**Examples**

```
>>> apiobj = self.{risksenseobject}.networks.update(291935, 'test_aug1')
```

**Note:** You can also dump the data of the network job id in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.networks.update(291935, 'test_aug1',csvdump=True)
```

**delete**(*network_id*, *client_id=None*, *csvdump=False*)

Deletes a network.

**Parameters**

- **network_id** (int) – The network ID to be deleted.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
True/False indicating whether or not the operation was successful.

**Examples**

```
>>> apiobj = self.[risksenseobject].networks.delete(1234,client_id=123)
```

**Note:** You can also dump the data of the network tht will be deleted in a csv file. Just make csvdump as True:

```
>>>  self.[risksenseobject].networks.delete(1234,client_id=123,csvdump=True)
```

**get_single_search_page**(*search_filters*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns networks based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_num** (int) – Page number of results to be returned.
- **page_size** (int) – Number of results to be returned per page.
- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
A paginated JSON response from the platform is returned.

**Examples**

```
>>> apiobj = self.[risksenseobject].networks.get_single_search_page([{"field"
↪"name","exclusive":False,"operator":"IN","orWithPrevious":False,
↪"implicitFilters":[],"value":"test"}])
```

**search**(*search_filters*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*, *csvdump=False*)

Searches for and returns networks based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_size** (int) – The number of results per page to be returned.

- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Whether retrieved search details to be dumped in a csv file or not

**Return type**
list

**Returns**
A list containing all networks returned by the search using the filter provided.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.networks.search([{"field":"name","exclusive
↪": False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"test"}])
```

---

**Note:** You can also dump the data of the network search in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.networks.search([{"field":"name","exclusive":False,
↪"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":"test"}],
↪csvdump=True)
```

---

**get_model**(*client_id=None*)

Get available projections and models for Networks.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Networks projections and models are returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.networks.get_model()
```

---

**suggest**(*search_filters*, *suggest_filter*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filters** (list) – Active Filters input
- **suggest_filter** (dict) – Suggest Filter input
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Value suggestions

**Examples**

```
>>> apiobj = self.{risksenseobject}.networks.suggest([],{"field":"name",
↪"exclusive":False,"operator":"WILDCARD","value":"tes*","implicitFilters":[]})
```

### 1.2.20 Roles (risksense_api.__subject.__role.__role)

Role module defined for different role related api endpoints.

**class** risksense_api.__subject.__role.__role.**Role**(*profile*)

Bases: Subject

Class for role function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise role function:

**Usage:**
self.{risksenseobjectname}.roles.{function}

**Examples**

To create a role using *create()* function

```
>>> self.rs.roles.create(args)
```

**__init__**(*profile*)

Initialization of Roles object.

**Parameters**
**profile** (object) – Profile Object

**create**(*name*, *description*, *client_id=None*)

Create a new role.

**Parameters**

- **name** (str) – The name for the new network.
- **decription** – The description of the role.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The new role ID.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.create('test','test')
```

**Parameters**
**description** (str) –

**allow_privileges**(*roleid*, *client_id=None*)

Allow privileges to a role.

**Parameters**

- **roleid** (int) – Rold Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Job State

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.allow_privileges(1234)
```

**deny_privileges**(*roleid*, *client_id=None*)

Deny privileges to a role.

**Parameters**

- **roleid** (int) – Rold Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Job State

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.deny_privileges(1234)
```

**delete_privileges**(*roleid*, *client_id=None*)

Delete privileges to a role.

**Parameters**

- **roleid** (int) – Rold Id

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Job State

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.delete_privileges(1234)
```

get_privileges(*client_id=None*)

Get privileges to a role.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
tuple

**Returns**
(Privilege Id, Privilege Name)

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.get_privileges()
```

update(*roleid*, *name*, *description*, *client_id=None*)

Update a new role.

**Parameters**

- **roleid** (int) – Rold Id
- **name** (str) – The name for the new role.
- **description** (str) – The description of the role.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The new role ID.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.update(123, 'test', 'test')
```

**delete_role**(*roleid*, *client_id=None*)

Update a new role.

**Parameters**

- **roleid** (int) – Rold Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
State(True/False)

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.delete_role('xxxx')
```

### 1.2.21 Users (risksense_api.__subject.__users.__users)

User module defined for different user related api endpoints.

**class** risksense_api.__subject.__users.__users.**Users**(*profile*)

Bases: Subject

Class for user function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise user function:

**Usage:**
self.{risksenseobjectname}.users.{function}

**Examples**

To create an user using *create()* function

```
>>> self.{risksenseobjectname}.users.create(args)
```

**__init__**(*profile*)

Initialization of Users object.

**Parameters**
**profile** (object) – Profile Object

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

Download user data based on search filters.

**Parameters**

- **filename** (str) – Name of the file
- **filters** (list) – A list of dictionaries containing filter parameters

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

---

**Note:** **IGNORE** - Internal funtion for csv dump

---

**remove_users**(*useruuid*, *client_id=None*, *csvdump=False*)

Delete a user

**Parameters**

- **useruuid** (str) – User UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
Job Id

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.remove_users('123-456')
```

---

**Note:** You can also dump the data of the group search in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.users.remove_users('123-456',csvdump=True)
```

---

**get_user_iaminfo**(*useruuid*, *client_id=None*, *csvdump=False*)

Get IAM info of a user.

**Parameters**

- **useruuid** (str) – User UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
Json response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.get_user_iaminfo('123-456')
```

**Note:** You can also dump the data of the group search in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.users.get_user_iaminfo('123-456',csvdump=True)
```

**assign_group**(*filter*, *targetgroupids*, *client_id=None*, *csvdump=False*)

Assign users to groups.

**Parameters**

- **filter** (list) – Search Filter
- **targetgroupids** (list) – Target group ids
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
Json response

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.assign_group([{"field":"id","exclusive
↪" False,"operator":"EXACT","value":"6506","implicitFilters":[]}],[1234])
```

**Note:** You can also dump the data of the group search in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.users.assign_group([{"field":"id","exclusive":False,
↪"operator":"EXACT","value":"6506","implicitFilters":[]}],[1234],csvdump=True)
```

**unassign_group**(*filter*, *targetgroupids*, *client_id=None*, *csvdump=False*)

Unassign users to groups.

**Parameters**

- **filter** (list) – Search Filter
- **targetgroupids** (list) – Target group ids
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
Json response

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.unassign_group([{"field":"id",
↪"exclusive":False,"operator":"EXACT","value":"6506","implicitFilters":[]}],
↪[1234])
```

**Note:** You can also dump the data of the group search in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.users.unassign_group([{"field":"id","exclusive
↪":False,"operator":"EXACT","value":"6506","implicitFilters":[]}],[1234],
↪csvdump=True)
```

**get_my_profile**(*csvdump=False*)

Get the profile for the user that owns the API key being used.

**Parameters**
**csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
A dictionary containing the user's profile.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.get_my_profile()
```

**Note:** You can also dump the data of the user profile in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.users.get_my_profile(csvdump=True)
```

**disallow_tokens**(*client_ids*, *user_id*)

Disallow use of tokens for a user.

**Parameters**

- **client_ids** (list) – List of client Ids
- **user_id** (int) – The ID of the user to be disallowed from token use.

**Return type**
bool

**Returns**
True/False indicating success or failure of submission of the operation.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.disallow_tokens([123,456],1234)
```

---

**allow_tokens**(*client_ids*, *user_id*)

Allow use of tokens for a user.

**Parameters**

- **client_ids** (list) – List of client Ids
- **user_id** (int) – The ID of the user to be disallowed from token use.

**Return type**
bool

**Returns**
True/False indicating success or failure of submission of the operation.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.allow_tokens([123,456],1234)
```

---

**getexporttemplate**(*client_id=None*)

Gets configurable export template for application findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.getexporttemplate()
```

---

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*)

Initiates an export job on the platform for user based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Possible options : ExportRowNumbers.ROW_5000, ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.XML, or ExportFileType.XLSX

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID in the platform from is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.export([{"field":"id","exclusive
↪":False,"operator":"EXACT","value":"6506","implicitFilters":[]}])
```

**get_single_search_page**(*search_filters*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns users based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_num** (int) – Page number of results to be returned.
- **page_size** (int) – Number of results to be returned per page.
- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Paginated JSON response from the platform.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.get_single_search_page([{"field":"id",
↪"exclusive":False,"operator":"EXACT","value":"6506","implicitFilters":[]}])
```

**search**(*search_filters*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns users based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **csvdump** (bool) – Toggle to dump data in csv

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all hosts returned by the search using the filter provided.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.search([{"field":"id","exclusive
↪":False,"operator":"EXACT","value":"6506","implicitFilters":[]}])
```

**Note:** You can also dump the data of the users in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.users.search([{"field":"id","exclusive":False,
↪"operator":"EXACT","value":"6506","implicitFilters":[]}],csvdump=True)
```

**list_user_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**

- **filter_subject** – Supported Subjects are:
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON output from the platform is returned, listing the available filters.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.list_user_filter_fields()
```

**get_user_info**(*user_id=None*, *client_id=None*, *csvdump=False*)

Get info for a specific user. If user_id is not specified, the info for the requesting user is returned.

**Parameters**

- **user_id** (typing.Optional[int]) – User ID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
User information.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.get_user_info(1234)
```

**Note:** You can also dump the data of the user information in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.users.get_user_info(1234,csvdump=True)
```

`create`(*username*, *first_name*, *last_name*, *email_address*, *group_ids=[]*, *client_id=None*, *read_only=False*, *csvdump=False*, ***kwargs*)

Create a new user.

**Parameters**

- **username** (str) – Username
- **first_name** (str) – First Name
- **last_name** (str) – Last Name
- **email_address** (str) – E-mail address
- **group_ids** (list) – Group IDs to assign user to
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **read_only** (bool) – Read only
- **csvdump** – Toggle to dump data in csv

**Keyword Arguments**

- **use_saml** (bool) – Is a SAML user?
- **saml_attr_1** (str) – SAML Attribute 1
- **saml_attr_2** (str) – SAML Attribute 2
- **exp_date** (str) – Expiration Date YYYY-MM-DD

**Return type**
int

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.create('test', 'test', 'test',
↪'abc@xyz.com')
```

**Note:** You can also dump the data of the user job id in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.users.create('test', 'test', 'test', 'abc@xyz.com',
↪csvdump=True)
```

**update_user_role**(*newrole*, *newexpiration*, *user_uuid*, *client_id=None*, *csvdump=False*)

Update user role.

**Parameters**

- **newrole** (str) – New User role id
- **newexpiration** (datetime.datetime) – Expiration date. Allowed format : "YYYY-MM-DDTHH:MM:SSZ"(eg: 2020-12-31T00:00:00.000Z)
- **user_uuid** (str) – User UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
Roles JSON

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.update_user_role('test','2020-12-
↪31T00:00:00.000Z','123-456')
```

**Note:** You can also dump the data of the user profile in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.users.update_user_role('test','2020-12-31T00:00:00.
↪000Z','123-456',csvdump=True)
```

**update_user**(*user_uuid*, *client_id=None*, *csvdump=False*, ***kwargs*)

Update a user.

**Parameters**

- **user_uuid** (str) – User UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Keyword Arguments**

- **username** (str) – Username
- **first_name** (str) – First Name
- **last_name** (str) – Last Name
- **email** (str) – Email
- **phone** (str) – Phone Num.
- **group_ids** (list) – Group IDs

- **read_only** (bool) – Read-Only
- **use_saml** (bool) – Use SAML?
- **saml_attr_1** (str) – SAML Attribute 1
- **saml_attr_2** (str) – SAML Attribute 2
- **exp_date** (str) – Expiration Date YYYY-MM-DD

**Return type**
int

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.update_user('123-456',username='test',
↪first_name='test')
```

**Note:** You can also dump the data of the user in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.users.update_user('123-456',username='test',first_
↪name='test',csvdump=True)
```

**send_welcome_email**(*search_filter*, *client_id=None*)

Send welcome e-mail to users identified by the search filter(s) provided.

**Parameters**

- **search_filter** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.send_welcome_email([{"field":"id",
↪"exclusive":False,"operator":"EXACT","value":"6506","implicitFilters":[]}])
```

**get_roles**(*client_id=None*)

Get roles

**Parameters**

- **search_filter** – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.[risksenseobject].users.get_roles()
```

**assign_clients**(*searchfilter*, *expirationdate*, *replacexistingroles=False*, *assignallgroups=False*, *client_id=None*, *client_idtouse=None*)

Assign user to clients

**Parameters**

- **searchfilter** (list) – A list of dictionaries containing filter parameters.
- **expirationdate** (str) – Expiration date. YYYY-mm-dd
- **replacexistingroles** (bool) – Replace existing roles
- **assignallgroups** (bool) – Assign all groups
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **client_idtouse** (typing.Optional[int]) – Client Id to use

**Return type**
int

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.[risksenseobject].users.assign_clients([{"field":"id",
↪"exclusive":False,"operator":"EXACT","value":"6506","implicitFilters":[]}],
↪'2022-01-01',client_idtouse=111)
```

**assign_roles**(*userid*, *expirationdate=None*, *replacexistingroles=False*, *assignallgroups=False*, *client_id=None*, *client_idtouse=None*)

Assign roles to user

**Parameters**

- **userid** (int) – User Id
- **expirationdate** (typing.Optional[str]) – Expiration date. YYYY-mm-dd
- **replacexistingroles** (bool) – Replace existing roles
- **assignallgroups** (bool) – Assign all groups
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **client_idtouse** (typing.Optional[int]) – Client Id to use

**Return type**
dict

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.assign_roles(123,'2022-01-01',client_
↪idtouse=111)
```

**remove_roles**(*userid*, *client_idtouse=None*, *client_id=None*)

Remove roles to user

**Parameters**

- **userid** (int) – User Id
- **client_idtouse** (typing.Optional[int]) – Client Id to use
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.remove_roles(123,client_idtouse=111)
```

**clientrolefiltering**(*client_ids=None*, *rolelabels=None*, *client_id=None*)

Client role filtering

**Parameters**

- **client_ids** (typing.Optional[list]) – Client Ids
- **rolelabels** (typing.Optional[list]) – Role Labels
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Job ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.clientrolefiltering(client_ids=[123,
↪12],rolelabels=['abc'])
```

**get_model**(*client_id=None*)

Get available projections and models for Users.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Users projections and models are returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.get_model()
```

**suggest**(*search_filter*, *suggest_filter*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter** (list) – Search Filter
- **suggest_filter** (dict) – Suggest Filter
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Value suggestions

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.suggest([],{"field":"id","exclusive
↪":False,"operator":"WILDCARD","value":"65*","implicitFilters":[]})
```

**import_users_csv**(*file_name*, *absolute_path_file*, *client_id=None*)

Add a file to an upload.

**Parameters**

- **file_name** (str) – The name to be used for the uploaded file.
- **absolute_path_file** (str) – Absolute path of the file to be uploaded
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The file ID is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.import_users_csv('test','C:\test.csv')
```

**systemuser_get_single_search_page**(*search_filters*, *page_num=0*, *page_size=150*, *sort_field='username'*, *sort_dir='DESC'*, *client_id=None*)

Searches for and returns users based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **page_num** (int) – Page number of results to be returned.
- **page_size** (int) – Number of results to be returned per page.
- **sort_field** (str) – Name of field to sort results on.
- **sort_dir** (str) – Direction to sort. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Paginated JSON response from the platform.

**Examples**

```
>>> apiobj = self.{risksenseobject}.users.systemuser_get_single_search_page([{
↪"field":"id","exclusive":False,"operator":"EXACT","value":"6506",
↪"implicitFilters":[]}])
```

### 1.2.22 Groups (risksense_api.__subject.__groups.__groups)

Group module defined for different group related api endpoints.

**class** risksense_api.__subject.__groups.__groups.**Groups**(*profile*)

Bases: Subject

Class for group function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise group function:

**Usage:**
self.{risksenseobjectname}.groups.{function}

**Examples**

To create a group using *create()* function

```
>>> self.{risksenseobjectname}.groups.create(args)
```

---

**__init__**(*profile*)

Initialization of Groups object.

**Parameters**
**profile** (object) – Profile Object

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

Download group data based on search filters.

**Parameters**

- **filename** (str) – Name of the file
- **filters** (list) – A list of dictionaries containing filter parameters
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

---

**Note:** **IGNORE** - Internal funtion for csv dump

---

**list_group_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON output from the platform is returned, listing the available filters.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.list_group_filter_fields()
```

---

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*, *csvdump=False*)

Searches for and returns groups based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to use
- **page_num** (int) – The page number of results to be returned.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting the results returned.
- **sort_dir** (str) – The direction of sorting to be used. (SortDirection.ASC or SortDirection.DESC)

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.get_single_search_page([{"field":
↪"name","exclusive":False,"operator":"WILDCARD","value":"test*",
↪"implicitFilters":[]}])
```

**Note:** You can also dump the data of the group search in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.groups.get_single_search_page([{"field":"name",
↪"exclusive":False,"operator":"WILDCARD","value":"test*","implicitFilters":[]}
↪],csvdump=True)
```

**search**(*search_filters*, *projection='detail'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*, *csvdump=False*)

Searches for and returns groups based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to use
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (int) – The field to be used for sorting the results returned.
- **sort_dir** (int) – The direction of sorting to be used. (SortDirection.ASC or SortDirection.DESC)
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
list

**Returns**
A list containing all host findings returned by the search using the filter provided.

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.search([{"field":"name","exclusive
↪":False,"operator":"WILDCARD","value":"test*","implicitFilters":[]}])
```

**Note:** You can also dump the data of the group search in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.groups.search([{"field":"name","exclusive":False,
↪"operator":"WILDCARD","value":"test*","implicitFilters":[]}],csvdump=True)
```

**create**(*name*, *description*, *client_id=None*, *csvdump=False*)

Creates a new group.

**Parameters**

- **name** (str) – The name to be used for the new group.
- **description** (str) – Group creation description
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
The new group ID is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.create('test','test')
```

**Note:** You can also dump the data of the group job id in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.groups.create('test','test',csvdump=True)
```

**history**(*groupid*, *client_id=None*, *csvdump=False*)

Get group history.

**Parameters**

- **groupid** (int) – The id to be used to fetch group history.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
dict

**Returns**
The history of group id.

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.history(1234)
```

**Note:** You can also dump the data of the group history in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.groups.history(1234,csvdump=True)
```

**delete**(*search_filters*, *client_id=None*, *csvdump=False*)

Deletes groups as specified in search_filters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
Job Id

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.delete([{"field":"name","exclusive
↪":False,"operator":"WILDCARD","value":"test*","implicitFilters":[]}])
```

**Note:** You can also dump the data of the group to be deleted in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.groups.get_single_search_page([{"field":"name",
↪"exclusive":False,"operator":"WILDCARD","value":"test*","implicitFilters":[]}
↪],csvdump=True)
```

**update_single_group**(*group_id*, *client_id=None*, *csvdump=False*, ***kwargs*)

Updates a group name and/or asset criticality.

**Parameters**

- **group_id** (int) – The group ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

Keyword Args: name (*str*): The new name. description (*str*): The new Description

**Return type**
int

**Returns**
The job ID is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.update_single_group(1234)
```

**Note:** You can also dump the data of the group job id in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.groups.update_single_group(1234,csvdump=True)
```

**assign**(*search_filters*, *user_ids*, *client_id=None*, *csvdump=False*)

Assign group(s) to user IDs, based on specified filter(s)

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **user_ids** (list) – A list of user IDs.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
The job ID is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.assign([{"field":"name","exclusive
↪" False,"operator":"EXACT","value":"test","implicitFilters":[]}],1234)
```

**Note:** You can also dump the data of the group job id in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.groups.assign([{"field":"name","exclusive":False
↪"operator":"EXACT","value":"test","implicitFilters":[]}],1234,csvdump=True)
```

**unassign**(*search_filters*, *user_ids*, *client_id=None*, *csvdump=False*)

Unassign group(s) from user IDs, based on specified filter(s)

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **user_ids** (list) – A list of user IDs.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
`int`

**Returns**
The job ID is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.unassign([{"field":"name","exclusive
↪":False,"operator":"EXACT","value":"test","implicitFilters":[]}],[1234])
```

---

**Note:** You can also dump the data of the group job id in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.groups.unassign([{"field":"name","exclusive":False,
↪"operator":"EXACT","value":"test","implicitFilters":[]}],[1234],csvdump=True)
```

---

`get_model`(*client_id=None*)

Get available projections and models for Groups.

**Parameters**
**`client_id`** (`typing.Optional[int]`) – Client ID

**Return type**
`dict`

**Returns**
Group projections and models are returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.get_model()
```

---

`suggest`(*search_filter*, *suggest_filter*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **`search_filter`** (`list`) – Search Filter
- **`suggest_filter`** (`dict`) – Suggest Filter
- **`client_id`** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`dict`

**Returns**
Value suggestions

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.suggest([{"field":"name","exclusive
↪":False,"operator":"WILDCARD","value":"test*","implicitFilters":[]}])
```

**getexporttemplate**(*client_id=None*)

Gets configurable export template for application findings.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.getexporttemplate()
```

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*, *csvdump=False*)

Initiates an export job on the platform for group(s) based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Possible options : ExportRowNumbers.ROW_5000, ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000", ExportRowNumbers.ROW_100000", ExportRowNumbers.ROW_ALL
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.XML, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.
- **csvdump** (bool) – Toggle to dump data in csv

**Return type**
int

**Returns**
The Export job ID in the platform from is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.export([{"field":"name","exclusive
↪":False,"operator":"WILDCARD","value":"test*","implicitFilters":[]}],'test')
```

**Note:** You can also dump the data of the group in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.groups.export([{"field":"name","exclusive":False,
↪"operator":"WILDCARD","value":"test*","implicitFilters":[]}],csvdump=True)
```

**subscribe_change_in_grouprs3**(*client_id=None*)

Subscribe change in group rs3 notification

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.subscribe_change_in_grouprs3()
```

**unsubscribe_change_in_grouprs3**(*client_id=None*)

Unsubscribe change in group rs3 notification

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.groups.unsubscribe_change_in_grouprs3()
```

### 1.2.23 Vulnerabilities (risksense_api.__subject.__vulnerabilities.__vulnerabilities)

Vulnerability module defined for different vulnerability related api endpoints.

**class** risksense_api.__subject.__vulnerabilities.__vulnerabilities.**Vulnerabilities**(*profile*)

Bases: Subject

Class for vulnerability function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise vulnerability function:

**Usage:**
self.{risksenseobjectname}.vulnerabilities.{function}

**Examples**

To get model for vulnerability using *get_model()* function

```
>>> self.{risksenseobjectname}.vulnerabilities.get_model()
```

**__init__**(*profile*)

Initialization of Vulnerabilities object.

**Parameters**
**profile** (object) – Profile Object

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns vulnerabilities based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all vulnerability returned by the search using the filter provided.

**Examples**

```
>>> apiobj = self.{risksenseobject}.vulnerabilities.search([{"field":"vrrGroup",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"Critical"}])
```

**Note:** You can also dump the data of the vulnerability search in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.vulnerabilities.search([{"field":"vrrGroup",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"Critical"}],csvdump=True)
```

---

**list_vulnerability_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON output from the platform is returned, listing the available filters.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.vulnerabilities.list_vulnerability_filter_
↪fields(client_id=123)
```

---

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns vulnerabilities based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_num** (int) – The page number of results to be returned.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.vulnerabilities.get_single_search_page([{
↪"field":"vrrGroup","exclusive":False,"operator":"IN","orWithPrevious":False,
↪"implicitFilters":[],"value":"Critical"}])
```

---

**get_model**(*client_id=None*)

Get available projections and models for Vulnerabilities.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Vulnerability projections and models are returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.vulnerabilities.get_model()
```

**suggest**(*search_filter*, *suggest_filter*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter** (list) – Search Filter
- **suggest_filter** (dict) – Suggest Filter
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Value suggestions

**Examples**

```
>>> apiobj = self.{risksenseobject}.vulnerabilities.suggest([],{"field"
↪"vrrGroup","exclusive":False,"operator":"WILDCARD","orWithPrevious":False
↪"implicitFilters":[],"value":"Critic*"})
```

**getexporttemplate**(*client_id=None*)

Gets configurable export template for Vulnerabilities.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
The Exportable fields

**Examples**

```
>>> apiobj = self.{risksenseobject}.vulnerabilities.getexporttemplate()
```

**export**(*search_filters*, *file_name*, *row_count='All'*, *file_type='CSV'*, *client_id=None*)

Initiates an export job on the platform for Vulnerabilities based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The name to be used for the exported file.
- **row_count** (str) – No of rows to be exported. Possible options : ExportRowNumbers.ROW_5000,ExportRowNumbers.ROW_10000, ExportRowNumbers.ROW_25000, ExportRowNumbers.ROW_50000, ExportRowNumbers.ROW_100000, ExportRowNumbers.ROW_ALL
- **file_type** (str) – File type to export. ExportFileType.CSV, ExportFileType.JSON, or ExportFileType.XLSX
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**

int

**Returns**

The job ID in the platform from is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.vulnerabilities.export([{"field":"vrrGroup",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"Critical"}],'test')
```

### 1.2.24 Weaknesses (risksense_api.__subject.__weaknesses.__weaknesses)

Weakness module defined for different weakness related api endpoints.

**class** risksense_api.__subject.__weaknesses.__weaknesses.**Weaknesses**(*profile*)

Bases: Subject

Class for weakness function definitions.

**Parameters**

**profile** (object) – Profile Object

To utlise weakness function:

**Usage:**

self.{risksenseobjectname}.weaknesses.{function}

**Examples**

To get model for weakness using *get_model()* function

```
>>> self.{risksenseobjectname}.weaknesses.get_model()
```

**__init__**(*profile*)

Initialization of Weaknesses object.

**Parameters**
**profile** (object) – Profile Object

**downloadfilterinexport**(*filename*, *filters*, *client_id=None*)

Download weakness data based on search filters.

**Parameters**

- **filename** (str) – Name of the file
- **filters** (list) – A list of dictionaries containing filter parameters
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

---

**Note:** **IGNORE** - Internal funtion for csv dump

---

**search**(*search_filters*, *projection='basic'*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *csvdump=False*, *client_id=None*)

Searches for and returns weaknesses based on the provided filter(s) and other parameters. Rather than returning paginated results, this function cycles through all pages of results and returns them all in a single list.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
list

**Returns**
A list containing all weaknesses returned by the search using the filter provided.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.weaknesses.search([{"field":"vulnId",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"CWE-918"}])
```

---

---

**Note:** You can also dump the data of the vulnerability search in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.weaknesses.search([{"field":"vulnId","exclusive
↪":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],"value":
↪"CWE-918"}],csvdump=True)
```

---

**list_weakness_filter_fields**(*client_id=None*)

List filter endpoints.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON output from the platform is returned, listing the available filters.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.weaknesses.list_weakness_filter_fields()
```

---

**get_single_search_page**(*search_filters*, *projection='basic'*, *page_num=0*, *page_size=150*, *sort_field='id'*, *sort_dir='ASC'*, *client_id=None*)

Searches for and returns weaknesses based on the provided filter(s) and other parameters.

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **projection** (str) – Projection to be used in API request. Projection.BASIC or Projection.DETAIL
- **page_num** (int) – The page number of results to be returned.
- **page_size** (int) – The number of results per page to be returned.
- **sort_field** (str) – The field to be used for sorting results returned.
- **sort_dir** (str) – The direction of sorting to be used. SortDirection.ASC or SortDirection.DESC
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.weaknesses.get_single_search_page([{"field":
↪"vulnId","exclusive":False,"operator":"IN","orWithPrevious":False,
↪"implicitFilters":[],"value":"CWE-918"}])
```

---

**get_model**(*client_id=None*)

Get available projections and models for weaknesses.

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Weaknesses projections and models are returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.weaknesses.get_model()
```

**suggest**(*search_filter*, *suggest_filter*, *client_id=None*)

Suggest values for filter fields.

**Parameters**

- **search_filter** (list) – Search Filter
- **suggest_filter** (dict) – Suggest Filter
- **client_id** (typing.Optional[int]) – Client ID

**Return type**
dict

**Returns**
Value suggestions

**Examples**

```
>>> apiobj = self.{risksenseobject}.weaknesses.suggest([],{"field":"vulnId",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"CWE*"})
```

**get_export_template**(*client_id=None*)

Get all fields that are part of configurable export

**Parameters**
**client_id** (typing.Optional[int]) – Client ID

**Return type**
list

**Returns**
Fields that can be configured for export

**Examples**

```
>>> apiobj = self.{risksenseobject}.weaknesses.get_export_template()
```

**export**(*search_filters*, *file_name*, *file_type='CSV'*, *row_count='All'*, *comment=''*, *client_id=None*)

Initiates an export job on the platform for weaknesses based on the provided filter(s).

**Parameters**

- **search_filters** (list) – A list of dictionaries containing filter parameters.
- **file_name** (str) – The file name to be assigned to the export.
- **file_type** – The file type for the export. Options are ExportFileType.CSV, ExportFileType.JSON, and ExportFileType.XLSX
- **comment** – Any comment wished to be associated with the export.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The job ID is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.weaknesses.export([{"field":"vulnId",
↪"exclusive":False,"operator":"IN","orWithPrevious":False,"implicitFilters":[],
↪"value":"CWE-918"}],'test')
```

### 1.2.25 Uploads (risksense_api.__subject.__uploads.__uploads)

Upload module defined for different upload related api endpoints.

**class** risksense_api.__subject.__uploads.__uploads.**Uploads**(*profile*)

Bases: Subject

Class for upload function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise upload function:

**Usage:**
self.{risksenseobjectname}.uploads.{function}

**Examples**

Create an upload using *create()* function

```
>>> self.{risksenseobjectname}.uploads.create(args)
```

**__init__**(*profile*)

Initialization of Uploads object.

**Parameters**
**profile** (object) – Profile Object

**get_uploads**(*assessment_id*, *page_num=0*, *page_size=150*, *client_id=None*)

Get uploads associated with an assessment.

**Parameters**

- **assessment_id** (int) – The assessment ID.
- **page_num** (int) – The page number of results to return.
- **page_size** (int) – The number of results per page to return.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.get_uploads(1234,page_num=1,page_
↪size=100)
```

---

**create**(*name*, *assessment_id*, *network_id*, *client_id=None*)

Create a new upload.

**Parameters**

- **name** (str) – The name to be associated with the upload.
- **assessment_id** (int) – The assessment ID.
- **network_id** (int) – The network ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
int

**Returns**
The Upload ID

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.create('test',123,123,client_id=123)
```

---

**check_state**(*upload_id*, *client_id=None*)

Check the state of an upload.

**Parameters**

- **upload_id** (int) – The upload ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`str`

**Returns**
The current state of the upload is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.check_state(1234)
```

---

**update**(*upload_id*, *name*, *network_id*, *assessment_id*, *client_id=None*)

Update an upload. Uploads can only be updated before they have been processed.

**Parameters**

- **upload_id** (`int`) – The upload ID.
- **name** (`str`) – File name
- **network_id** (`int`) – Network ID.
- **assessment_id** (`int`) – Assessment ID
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`int`

**Returns**
The upload ID is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.update(1234 'test' 123 123 client_
↪id=123)
```

---

**delete**(*upload_id*, *client_id=None*)

Delete an Upload.

**Parameters**

- **upload_id** (`int`) – The upload ID
- **client_id** (`typing.Optional[int]`) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
`bool`

**Returns**
True/False reflecting whether or not the operation was successful.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.delete(1234)
```

---

**list_files**(*upload_id*, *page_num=0*, *page_size=150*, *client_id=None*)

List files in an upload.

**Parameters**

- **upload_id** (int) – The upload ID
- **page_num** (int) – The page number to be returned.
- **page_size** (int) – The number of results to return per page.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
A paginated JSON response from the platform.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.list_files(1234,page_num=1,page_
↪size=100)
```

---

**add_file**(*upload_id*, *file_name*, *path_to_file*, *client_id=None*)

Add a file to an upload.

**Parameters**

- **upload_id** (int) – Upload ID
- **file_name** (str) – The name to be used for the uploaded file.
- **path_to_file** (str) – Full path to the file to be uploaded.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The file ID along with jsonified response is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.add_file(1234,'test','D:\test\test.
↪nessus')
```

---

**update_file**(*upload_id*, *file_id*, *client_id=None*, ***kwargs*)

Update an uploaded file. Will only work if the file has not yet been processed.

**Parameters**

- **upload_id** (int) – The upload ID.
- **file_id** (int) – The file ID.

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Keyword Arguments**

- **assessment_id** (int) – The assessment ID the upload should be associated with. Integer.
- **network_id** (int) – The network ID the upload should be associated with. Integer.
- **application_id** (int) – The application ID the upload should be associated with. Integer.

**Return type**
int

**Returns**
The upload ID is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.update_file(1234 123 assessment_
↪id=123,network_id=123,application_id=123)
```

---

**delete_file**(*upload_id*, *file_id*, *client_id=None*)

Delete an uploaded file.

**Parameters**

- **upload_id** (int) – The upload ID.
- **file_id** (int) – The file ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
True/False reflecting whether or not the operation was successfully submitted.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.delete_file(1234 123)
```

---

**download_file**(*upload_id*, *file_name*, *client_id=None*)

Download a previously uploaded file.

**Parameters**

- **upload_id** (int) – The upload ID
- **file_name** (str) – The filename
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
True/False reflecting whether or not the operation was successful.

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.download_file(1234 'test')
```

**fetch_file_by_uuid**(*upload_id*, *file_uuid*, *file_destination*, *client_id=None*)

Download a file by UUID.

**Parameters**

- **upload_id** (int) – The upload ID
- **file_uuid** (str) – The file UUID
- **file_destination** (str) – The local destination for the downloaded file.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
True/False reflecting whether or not the operation was successful.

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.fetch_file_by_uuid(1234 '123-456',
↪'D:\test\test.xml')
```

**start_processing**(*upload_id*, *auto_urba=False*, *client_id=None*)

Initiate processing of an upload.

**Parameters**

- **upload_id** (int) – The upload ID
- **auto_urba** (bool) – Indicator for whether or not auto-URBA should be run after upload is processed.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
True/False reflecting whether or not the operation was successfully submitted.

**Examples**

```
>>> apiobj = self.{risksenseobject}.uploads.start_processing(1234 True)
```

### 1.2.26 Notifications (risksense_api.__subject.__notifications.__notifications)

Notification module defined for different notification related api endpoints.

**class** risksense_api.__subject.__notifications.__notifications.**Notifications**(*profile*)

Bases: Subject

Class for notification function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise notification function:

**Usage:**
self.{risksenseobjectname}.notifications.{function}

---

**Examples**

To get a model for notification using *get_model()* function

```
>>> self.{risksenseobjectname}.notifications.get_model()
```

---

**__init__**(*profile*)

Initialization of Notifications object.

**Parameters**
**profile** (object) – Profile Object

**subscribe_notifications**(*notificationtypeid*, *csvdump=False*, *subscribe=True*, *client_id=None*)

Subscribe to a notification

**Parameters**

- **notificationtypeid** (int) – The notification id to subscribe.
- **csvdump** (bool) – dumps the data in csv
- **subscribe** – Whether to subscribe or not
- **client_id** – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Success json

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.subscribe_notifications(123)
```

---

---

**Note:** You can also dump the data of the notification in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.notifications.subscribe_notifications(123
↪csvdump=True)
```

---

**unsubscribe_notifications**(*notificationtypeid*, *csvdump=False*, *subscribe=False*, *client_id=None*)

Unsubscribe to a notification

**Parameters**

- **notificationtypeid** (int) – The notification id to subscribe.
- **subscribe** (bool) – Whether to subscribe or not
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Success json

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.unsubscribe_notifications(123)
```

---

**Note:** You can also dump the data of the notification in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.notifications.unsubscribe_notifications(123
↪csvdump=True)
```

---

**markasread**(*notificationids*, *markasread=False*, *client_id=None*)

Mark as read/unread notifications

**Parameters**

- **notificationtypeid** – The notification id to subscribe.
- **markasread** (bool) – Whether to markread or not
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
bool

**Returns**
Success json

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.markasread(123
↪markasread=True)
```

---

**Parameters**
**notificationids** (list) –

**create_delivery_channel**(*channelname*, *channeltype*, *address*, *verificationcode*, *webhookcontenttype=None*, *client_id=None*)

Creates delivery channel for the user

**Parameters**

- **channelname** (str) – Name of channel
- **channeltype** (str) – Type of channel
- **address** (str) – Address
- **verificationcode** (str) – Verification code of user
- **webhookcontenttype** (typing.Optional[bool]) – Webhook content type
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
str

**Returns**
Status

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.create_delivery_channel('test
↪','EMAIL','abc@xyz.com','12345')
```

---

**edit_delivery_channel**(*channelid*, *channelname*, *channeltype*, *address*, *verificationcode*, *webhookcontenttype=None*, *disabled=False*, *shared=False*, *client_id=None*)

Edits delivery channel for the user

**Parameters**

- **channelid** (int) – Channel id
- **channelname** (str) – Name of channel
- **channeltype** (str) – Type of channel
- **address** (str) – Address
- **verificationcode** (int) – Verification code of user
- **webhookcontenttype** – Webhook content type
- **disabled** (bool) – Enable/disable notifications
- **shared** (bool) – Shared
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
str

**Returns**
Status

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.edit_delivery_channel(123
↪'test','EMAIL','abc@xyz.com','1234')
```

---

**delete_delivery_channel**(*channelids*, *client_id=None*)

Deletes delivery channels

**Parameters**

- **channelids** (list) – Channel ids
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
str

**Returns**
Status

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.delete_delivery_channel([123
↪123])
```

---

**list_channel**(*order='ASC'*, *csvdump=False*, *client_id=None*)

Get list of channel for admin

**Parameters**

- **order** (str) – sort order
- **csvdump** (bool) – Toogle to dump the data to csv
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
List channel

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.list_channel()
```

---

---

**Note:** You can also dump the data of the notification in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.notifications.list_channel(csvdump=True)
```

---

**list_channel_user**(*order='ASC'*, *csvdump=False*, *client_id=None*)

List delivery channels for normal user

**Parameters**

- **order** (str) – Sort order
- **csvdump** (bool) – csvdump
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Status

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.list_channel_user()
```

**Note:** You can also dump the data of the notification in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.notifications.list_channel_user(csvdump=True)
```

**send_verification_code**(*channelname*, *channeladdress*, *channeltype*, *client_id=None*)

Sends verification code to the user

**Parameters**

- **channelname** (str) – Name of the channel.
- **channeladdress** (str) – Address of the channel.
- **channeltype** (str) – TYPE of channel
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
str

**Returns**
Status

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.send_verification_code('test',
↪'https://abc.com','SLACK')
```

**get_model**(*client_id=None*)

List projections and their models that can be requested from the search endpoint.

**Parameters**
**client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Model of endpoint

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.get_model()
```

`search_filters`(*client_id=None*)

List fields that can be filtered by in the search endpoint

**Parameters**
**client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Search filter fields

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.search_filters()
```

`notification_search`(*filters*, *csvdump=False*, *client_id=None*)

Search for notifications

**Parameters**

- **filters** (list) – Search filters
- **csvdump** (bool) – csvdump
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Success json

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.notification_search([])
```

**Note:** You can also dump the data of the notification in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.notifications.notification_search([],csvdump=True)
```

**enablenotification**(*id*, *channelname*, *channeltype*, *client_id=None*)

Enable notifications

**Parameters**

- **id** (int) – Notification Id
- **channelname** (str) – Channel name
- **channeltype** (str) – Channel type
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.enablenotification(123 'test',
↪'EMAIL')
```

---

**disablenotification**(*id*, *channelname*, *channeltype*, *client_id=None*)

Disable notifications

**Parameters**

- **id** (int) – Notification Id
- **channelname** (str) – Channel name
- **channeltype** (str) – Channel type
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.disablenotification(123 'test
↪', 'EMAIL')
```

---

**get_notifications**(*page=0*, *page_size=50*, *client_id=None*)

Get Notifications

**Parameters**

- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id
- **page** (int) – Page number

- **page_size** (int) – Page size

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.get_notifications(client_
↪id=123)
```

---

**get_detailpane**(*notification_id*, *csvdump=False*, *client_id=None*)

Get details pane

**Parameters**

- **notificationid** – The notiifcation id to get details of
- **csvdump** (bool) – dumps the data in csv
- **client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Return type**
list

**Returns**
Notifications detail

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.get_detailpane(123)
```

---

**Note:** You can also dump the data of the notification in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.notifications.subscribe_notifications(123
↪csvdump=True)
```

---

**Parameters**
**notification_id** (int) –

**get_delivery_channel_template**(*client_id=None*)

Get delivery channel template

**Parameters**
**client_id** (typing.Optional[int]) – The client id , if none will provide the default client id

**Returns**
Delivery channel template

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.get_delivery_channel_
↪template()
```

**trigger_systemfilter**(*filterid*, *subject*, *description*, *client_id=None*)

Trigger system filter

**Parameters**

- **filterid** (int) – Filter Id
- **subject** (str) – Filter subject
- **description** (str) – Description

**Returns**

Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.notifications.trigger_systemfilter(123 'test
↪' 'test')
```

**Parameters**

**client_id** (typing.Optional[int]) –

### 1.2.27 Connectors (risksense_api.__subject.__connectors.__connectors)

Connector module defined for different connector related api endpoints.

**class** risksense_api.__subject.__connectors.__connectors.**Connectors**(*profile*)

Bases: Subject

Class for connector function definitions.

**Parameters**

**profile** (object) – Profile Object

To utlise group function:

**Usage:**

self.{risksenseobjectname}.connectors.{function}

**Examples**

To update a connector using *update()* function

```
>>> self.{risksenseobjectname}.connectors.update(args)
```

**class Type**

Bases: object

Connectors.Type class

```
NESSUS = 'NESSUS'

QUALYS_VULN = 'QUALYS_VULN_FILE_PICKUP'

QUALYS_VMDR = 'QUALYS_API_VULNERABILITY'

QUALYS_ASSET = 'QUALYS_ASSET'

NEXPOSE = 'NEXPOSE_FILE_PICKUP'

TENEBLE_SEC_CENTER = 'TENEBLE_SECURITY_CENTER'

BURPSUITE = 'BURPSUITE'

CROWDSTRIKE = 'FALCONSPOTLIGHT'

QUALYS_WAS = 'QUALYS_WAS'

VERACODE = 'VERACODE'

SONAR_CLOUD = 'SONARCLOUD'

JIRA = 'JIRA'

SERVICENOW_INCIDENT = 'SERVICE_NOW'

SERVICENOW_SERVICEREQUEST = 'SNOW_SERVICE_REQUEST'

CHECKMARX_OSA = 'CHECKMARXOSA'

CHECKMARX_SAST = 'CHECKMARXSAST'

HCL_APPSCAN = 'HCL_ASOC'

QUALYS_PC = 'QUALYS_PC'

CHERWELL = 'CHERWELL'

SERVICENOW_CTC = 'GENERIC_SNOW'

AWSINSPECTOR = 'AWS_INSPECTOR'

EXPANDER = 'EXPANDER'

NEXPOSE_ASSET = 'NEXPOSE_ASSET_TAG_FILE_PICKUP'

PRISMA_CLOUD = 'PRISMACLOUD'

FORTIFY_ON_DEMAND = 'FORTIFYONDMD'

SONATYPE = 'SONATYPE'

AQUASEC = 'AQUASEC'

WHITEHAT = 'WHITEHAT'
```

**class ScheduleFreq**

Bases: `object`

Connectors.ScheduleFreq class

```
DAILY = 'DAILY'

WEEKLY = 'WEEKLY'

MONTHLY = 'MONTHLY'
```

**__init__**(*profile*)

Initialization of Connectors object.

**Parameters**
**profile** (object) – Profile Object

**get_single_connector**(*connector_id*, *csvdump=False*, *client_id=None*)

Get a connector detail based on connector id.

**Parameters**

- **connector_id** (int) – Connector Id
- **csvdump** (bool) – Whether to dump the assessment history in a csv, true to dump and false to not dump
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Examples**

```
>>> apiobj = self.[risksenseobject].connectors.get_single_connector(123,client_
↪id=123)
```

---

---

**Note:** You can also dump the data of the connector search in a csv file. Just make csvdump as True:

```
>>>  self.[risksenseobject].connectors.get_single_connector(123,client_id=123,
↪csvdump=True)
```

---

**get_snow_category**(*snow_username*, *snow_password_or_token*, *snow_url*, *catalogid*, *client_id=None*)

Get ServiceNow Category information.

**Parameters**

- **snow_username** (str) – Service Now username
- **snow_password_or_token** (str) – Service Now API Token/Password
- **snow_url** (str) – Service Now Platform URL
- **catalogid** (str) – Catalog id to get category of
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Jsonified response of category data

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_snow_category('test','xxx',
↪'https://test.com',1234,client_id=123)
```

---

**get_snow_item**(*snow_username*, *snow_password_or_token*, *snow_url*, *catalogid*, *categoryid*, *client_id=None*)

Get ServiceNow item data

**Parameters**

- **snow_username** (str) – Service Now username
- **snow_password_or_token** (str) – Service Now API Token/Password
- **snow_url** (str) – Service Now Platform URL
- **catalogid** (str) – Service Now catalog id
- **categoryid** (str) – Service Now category id
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Jsonified response of item data

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_snow_item('test','xxx',
↪'https://test.com',1234,123,client_id=123)
```

---

**get_snow_catalogitemvariables**(*snow_username*, *snow_password_or_token*, *snow_url*, *catalogid*, *categoryid*, *catalogitemid*, *client_id=None*)

Get ServiceNow fields item data from items

**Parameters**

- **snow_username** (str) – Service Now username
- **snow_password_or_token** (str) – Service Now API Token/Password
- **snow_url** (str) – Service Now Platform URL
- **catalogid** (str) – Service Now catalog id
- **categoryid** (str) – Service Now category id
- **catalogitemid** (str) – Service Now item id
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Jsonified response of item data

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_snow_catalogitemvariables(
↪'test', 'xxx', 'https://test.com', 1234, 123, 123, client_id=123)
```

`run_connector`(*connectorid*, *client_id=None*)

Run connector based on connector id

**Parameters**

- **connectorid** (int) – Connector Id
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Json response of connector run

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.run_connector(123, client_id=123)
```

`paginate_connector`(*page_num=0*, *page_size=150*, *csvdump=False*, *client_id=None*)

Get a paginated list of connectors associated with the client.

**Parameters**

- **page_num** (int) – The page number of results to be returned
- **page_size** (int) – The number of results to return per page
- **csvdump** (bool) – Whether to dump the connector data in a csv, true to dump and false to not dump
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.paginate_connector(page_num=0,
↪page_size=100, client_id=123)
```

**Note:** You can also dump the data of the connector search in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.connectors.paginate_connector(page_num=0,page_
↪size=100,client_id=123,csvdump=True)
```

---

**connector_populate**(*body*, *client_id=None*)

Populate data of connector.

**Parameters**

- **body** (dict) – Populate body
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.connector_populate({"type":"JIRA
↪","username":"test@xyz.com","password":"test","url":"https://test.atlassian.
↪net","projection":"internal","connectorId":1234},client_id=123)
```

---

**Note:** Intercept this API endpoint `/client/{clientId}/connector/populate` in UI to better understand the body that need to be sent using this function. Then, use this function in your automation.

---

**create_scanning_connector**(*conn_name*, *conn_type*, *conn_url*, *schedule_freq*, *network_id*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new scanning connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_type** (str) – The connector type
- **conn_url** (str) – The URL for the connector to communicate with.
- **schedule_freq** (str) – The frequency for the connector to run. Options:
- **Connectors.ScheduleFreq.DAILY** –
- **Connectors.ScheduleFreq.WEEKLY** –
- **Connectors.ScheduleFreq.MONTHLY** –
- **network_id** (int) – The network ID
- **username_or_access_key** (str) – The username to use for connector authentication
- **password_or_secret_key** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **template** (str) – Template data
- **create_asset** (bool) – Create asset data
- **reportnameprefix** (str) – Report Name Prefix
- **authmechanism** (str) – Auth mechanism
- **ingestionfindingstype** (list) – Ingestion finding type
- **folder_id** (int) – Nessus scanner folder id

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_scanning_connector('test',
↪'JIRA','https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_
↪id=123,hour_of_day=0,folder_id=1)
```

**Note:** Intercept this API endpoint `/client/{clientId}/connector` in UI to better understand the argument values that need to be sent using this function. Then, use this function in your automation.

**create_expander**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *apikey*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new expander connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Option: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **apikey** (str) – The password/api token to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?

- **client_id** (typing.Optional[bool]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_expander('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

---

**Note:** Intercept this API endpoint /client/{clientId}/connector in UI to better understand the argument values that need to be sent using this function. Then, use this function in your automation.

---

**cherwellincident_connector_populate**(*body*, *client_id=None*)

Populates cherwell incident connector data based on body

**Parameters**

- **body** (dict) – Body for connector populate
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
The JSON response from the platform is returned

---

**Note:** Intercept this API endpoint /client/{clientId}/connector/populate/ticketType/Incident in UI to better understand the body that need to be sent using this function. Then, use this function in your automation.

---

**cherwellproblem_connector_populate**(*body*, *client_id=None*)

Populates cherwell problem connector data based on body

**Parameters**

- **body** (dict) – Body for connector populate
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
The JSON response from the platform is returned

---

**Note:** Intercept this API endpoint `/client/{clientId}/connector/populate/ticketType/Problem` in UI to better understand the `body` that need to be sent using this function. Then, use this function in your automation.

---

**cherwellmakerequest_connector_populate**(*body*, *client_id=None*)

Populates cherwell make request connector data based on body

**Parameters**

- **body** (dict) – Body for connector populate
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
The JSON response from the platform is returned

---

**Note:** Intercept this API endpoint `/client/{clientId}/connector/populate/ticketType/ChangeRequest` in UI to better understand the `body` that need to be sent using this function. Then, use this function in your automation.

---

**create_cherwell_incident_connector**(*cw_name*, *cw_username*, *cw_password*, *clientid_key*, *cw_url*, *autourba=False*, *client_id=None*)

Creates cherwell incident type connector

**Parameters**

- **cw_name** (str) – Cherwell connector name
- **cw_username** (str) – Cherwell connector username
- **cw_password** (str) – Cherwell connector password
- **clientid_key** (str) – Cherwell connector client id key
- **cw_url** (str) – Cherwell connector url
- **autourba** (bool) – Switch to enable auto urba
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
int

**Returns**
Id of connector

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_cherwell_incident_
↪connector('test','test','pass','xxxx','https://test.com',autourba=True,client_
↪id=123)
```

---

**create_cherwell_problem_connector**(*cw_name*, *cw_username*, *cw_password*, *clientid_key*, *cw_url*, *autourba=False*, *client_id=None*)

Creates cherwell problem typpe connector

**Parameters**

- **cw_name** (str) – Cherwell connector name
- **cw_username** (str) – Cherwell connector username
- **cw_password** (str) – Cherwell connector password
- **clientid_key** (str) – Cherwell connector client id key
- **cw_url** (str) – Cherwell connector url
- **autourba** (bool) – Switch to enable auto urba
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
int

**Returns**
Connector id

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_cherwell_problem_
↪connector('test','test','pass','xxxx','https://test.com',autourba=True,client_
↪id=123)
```

---

**create_cherwell_makerequest_connector**(*cw_name*, *cw_username*, *cw_password*, *clientid_key*, *cw_url*, *autourba=False*, *client_id=None*)

Creates cherwell change request type connector

**Parameters**

- **cw_name** (str) – Cherwell connector name
- **cw_username** (str) – Cherwell connector username
- **cw_password** (str) – Cherwell connector password
- **clientid_key** (str) – Cherwell connector client id key
- **cw_url** (str) – Cherwell connector url
- **autourba** (bool) – Switch to enable auto urba
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
int

**Returns**
Connector id

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_cherwell_makerequest_
↪connector('test','test','pass','xxxx','https://test.com',autourba=True,client_
↪id=123)
```

**create_qualyspc**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new qualys pc connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_qualyspc('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

**create_hclappscan**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new hcl appscan connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_hclappscan('test','https:/
↪/test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

---

**create_veracode**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *access_key*, *secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new veracode connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY

- **network_id** (int) – The network ID
- **access_key** (str) – The access key to use for connector authentication
- **secret_key** (str) – The secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_veracode('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

**create_sonar_cloud**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *access_key*, *secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new sonar cloud connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **access_key** (str) – The access key to use for connector authentication
- **secret_key** (str) – The secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.

- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.frisksenseobject.connectors.create_sonar_cloud('test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**create_nessus**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *access_key*, *secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Nessus connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **access_key** (str) – The access key to use for connector authentication
- **secret_key** (str) – The secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **folder_id** (int) – Nessus scanner folder id. Integer

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_nessus('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

**create_tenableio**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *access_key*, *secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new tenable io connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **access_key** (str) – The access key to use for connector authentication
- **secret_key** (str) – The secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_tenableio('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

**create_awsinspector**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new aws inspector connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_awsinspector('test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

---

**create_burpsuite**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *apikey*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new burpsuite connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **apikey** (str) – The api key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_burpsuite('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

---

**create_whitehat**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *apikey*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Whitehat sentinel dynamic connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **apikey** (str) – The api key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_burpsuite('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

**create_aquasec**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *apikey*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new burpsuite connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **apikey** (str) – The api key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_burpsuite('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

**create_crowdstrike**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new crowdstrike connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.risksenseobject.connectors.create_crowdstrike('test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

---

**create_snow_customtableconfig**(*conn_name*, *conn_url*, *username*, *password*, *tablename*, *statusfield*, *ticketidfield*, *enabletagremoval=False*, *enableuploadattachment=True*, *client_id=None*)

Create a new snow custom table configuration connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **tablename** (str) – Table name
- **statusfield** (str) – Status field
- **ticketidfield** (str) – Ticket Id field

- **enabletagremoval** (bool) – Enable Tag removal
- **enableuploadattachment** (bool) – Enable upload attachment
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_snow_customtableconfig(
↪'test','https://test.com','xxxx','xxxx','test','open','123',client_id=123)
```

---

**get_jira_project**(*username*, *password_or_api_token*, *jira_url*, *client_id=None*)

Get JIRA Projects

**Parameters**

- **username** (str) – JIRA username
- **password_or_api_token** (str) – JIRA API Token/Password
- **jira_url** (str) – JIRA Platform URL
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
tuple

**Returns**

- JIRA Project Name
- JIRA Project Key
- Description field enabled list

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_jira_project('test','test',
↪'https://test.com',client_id=123)
```

---

**get_jira_issuetype**(*username*, *password_or_api_token*, *jira_url*, *project_key*, *client_id=None*)

Get JIRA Issue Types

**Parameters**

- **username** (str) – JIRA username
- **password_or_api_token** (str) – JIRA API Token/Password
- **jira_url** (str) – JIRA Platform URL
- **project_key** (str) – JIRA Project Key
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
tuple

**Returns**

- JIRA Issue Type Name
- JIRA Issue Type Key

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_jira_issuetype('test','test',
↪'https://test.com','test',client_id=123)
```

---

**get_jira_issuetypefields**(*username*, *password_or_api_token*, *jira_url*, *project_key*, *issuetypeid*, *client_id=None*)

Get JIRA Issue Type Field key and value

**Parameters**

- **username** (str) – JIRA username
- **password_or_api_token** (str) – JIRA API Token/Password
- **jira_url** (str) – JIRA Platform URL
- **project_key** (str) – JIRA Project Key
- **issuetypeid** (str) – JIRA Issue type id
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
tuple

**Returns**

- JIRA Issue Type Name
- JIRA Issue Type Key

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_jira_issuetypefields('test',
↪'test','https://test.com','test',client_id=123)
```

---

**get_jira_tagtype_ticketstatus**(*username*, *password_or_api_token*, *jira_url*, *project_key*, *issue_type_key*, *client_id=None*)

Get JIRA Tag Type and Ticket Status options

**Parameters**

- **username** (str) – JIRA username
- **password_or_api_token** (str) – JIRA API Token/Password
- **jira_url** (str) – JIRA Platform URL
- **project_key** (str) – JIRA Project Key
- **issue_type_key** (str) – JIRA Issue Type Key

- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
tuple

**Returns**

- Tag Type Name
- JIRA Closed status name
- JIRA Closed status Key
- JIRA Ticket sync string

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_jira_issuetypefields('test',
↪'test','https://test.com','test','due_date',client_id=123)
```

---

**create_jira_connector**(*jira_connector_name*, *username*, *password_or_api_token*, *jira_url*, *project_name*, *project_key*, *issue_type_name*, *issue_type_key*, *tag_type_name*, *closed_status_key*, *closed_status_value*, *ticket_sync_string*, *supporteddescription*, *client_id=None*)

Create a JIRA Connector

**Parameters**

- **jira_connector_name** (str) – JIRA Connector Name
- **username** (str) – JIRA username
- **password_or_api_token** (str) – JIRA API Token/Password
- **jira_url** (str) – JIRA Platform URL
- **project_name** (str) – JIRA Project Name
- **project_key** (str) – JIRA Project Key
- **issue_type_name** (str) – JIRA Issue Type Name
- **issue_type_key** (str) – JIRA Issue Type Key
- **client_id** (typing.Optional[int]) – RS Client ID
- **tag_type_name** (str) – Tag Type Name
- **closed_status_value** (str) – JIRA Closed status name
- **closed_status_key** (str) – JIRA Closed status Key
- **ticket_sync_string** (str) – JIRA Ticket sync string
- **supporteddescription** (list) – Supported description

**Return type**
int

**Returns**
Created JIRA connector ID

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_jira_connector('test',
↪'test','test project','TP','bug','bug','CUSTOM','Closed','closed','Open,In␣
↪Progress','test',client_id=123)
```

---

**get_snow_fields**(*snow_username*, *snow_password_or_token*, *snow_url*, *client_id=None*)

Get Service Now Incident type connector for Tag Type and Ticket Status options

**Parameters**

- **snow_username** (str) – Service Now username
- **snow_password_or_token** (str) – Service Now API Token/Password
- **snow_url** (str) – Service Now Platform URL
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
tuple

**Returns**

- Tag Type Name
- Service Now Closed status name
- Service Now Closed status Key
- Service Now Ticket sync string

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_snow_fields('test','test',
↪'https://test.com','DAILY',client_id=123)
```

---

**get_snow_catalog**(*snow_username*, *snow_password_or_token*, *snow_url*, *client_id=None*)

Get Service Now connector data

**Parameters**

- **snow_username** (str) – Service Now username
- **snow_password_or_token** (str) – Service Now API Token/Password
- **snow_url** (str) – Service Now Platform URL
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Catalog,category,item information,close status key etc in populate data

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_snow_catalog('test','test',
↪'https://test.com',client_id=123)
```

---

**populate_editform_cmdb**(*filters*, *client_id=None*)

Populate editform cmdb data

**Parameters**

- **filters** (list) – Connector populate filters
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Jsonified response of editform populate data

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.populate_editform_cmdb([{"field
↪": "id","exclusive": False,"operator": "IN","value": "1223"}],client_id=123)
```

---

**populate_lockform_cmdb**(*filters*, *client_id=None*)

Populate lockform cmdb data

**Parameters**

- **filters** (str) – Connector populate filters
- **client_id** (typing.Optional[int]) – RS Client ID

**Returns**
Jsonified response of lockform populate data

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.populate_lockform_cmdb([{"field
↪": "id","exclusive": False,"operator": "IN","value": "1223"}],client_id=123)
```

---

**list_cmdb_custom_fields**(*client_id=None*)

Populate cmdb custom field data

**Parameters**
**client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> api = self.{risksenseobject}.connectors.list_cmdb_custom_fields(client_
↪id=123)
```

---

**add_cmdb_custom_fields**(*cmdbcustomfields*, *client_id=None*)

Add cmdb custom fields label

**Parameters**

- **cmdbcustomfields** (list) – Connector populate filters
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
bool

**Returns**
True/False sucess status

---

**Examples**

```
>>> api = self.{risksenseobject}.connectors.add_cmdb_custom_fields( [{"value":
↪"test","key": "cf_1"}],client_id=123)
```

---

**get_multiplematches**(*subject*, *assetid*, *client_id=None*)

Get list of assets that have multiple matches

**Parameters**

- **subject** (str) – Subject specified information of the asset id
- **assetid** (int) – Asset id to check for multiple matches
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Jsonified response of multiple matches

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_multiplematches('hostFinding
↪' 123,client_id=123)
```

---

**get_multiplematches_connector**(*subject*, *assetid*, *connectorid*, *hostid*, *sysid*, *table*, *client_id=None*)

Get list of assets that multiple matches of cmdb data

**Parameters**

- **subject** (str) – Subject specified information of the asset id
- **assetid** (int) – Asset id to check for multiple matches
- **connectorid** (int) – Connector id
- **hostid** (int) – Host id
- **sysid** (int) – Sys id
- **table** (str) – Table information

**Return type**
dict

**Returns**
Json data

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_multiplematches_connector(
↪'hostFinding' 123 123 123 123 'test',client_id=123)
```

**Parameters**
**client_id** (typing.Optional[int]) –

**create_asset_snowcmdb**(*subject*, *assetid*, *connectorid*, *client_id=None*)

Create an asset in snow cmdb

**Parameters**

- **subject** (str) – Subject specified information of the asset id
- **assetid** (int) – Asset id to check for multiple matches
- **connectorid** (int) – Connector id
- **client_id** (typing.Optional[int]) – RS Client ID,if none takes the default client id

**Return type**
dict

**Returns**
Json data

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_asset_snowcmdb(
↪'hostFinding' 123 123,client_id=123)
```

**sync_asset_snowcmdb**(*subject*, *assetid*, *connectorid*, *client_id=None*)

Sync asset in snow cmdb

**Parameters**

- **subject** (str) – Subject specified information of the asset id
- **assetid** (int) – Asset id to check for multiple matches
- **connectorid** (int) – Connector id
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
dict

**Returns**
Json data

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.sync_asset_snowcmdb('hostFinding
↪' 123 123,client_id=123)
```

**create_snow_connector**(*snow_connector_name*, *snow_username*, *snow_password_or_token*, *snow_url*, *tag_type_name*, *closed_status_value*, *closed_status_key*, *ticket_sync_string*, *supporteddescriptionfields*, *selected_optional_fields*, *client_id=None*)

Create a Service Now Incident type Connector

**Parameters**

- **snow_connector_name** (str) – SNOW Connector Name
- **snow_username** (str) – SNOW username
- **snow_password_or_token** (str) – SNOW API Token/Password
- **snow_url** (str) – SNOW Platform URL
- **tag_type_name** (str) – Tag Type Name
- **closed_status_value** (str) – SNOW Closed status name
- **closed_status_key** (str) – SNOW Closed status Key
- **ticket_sync_string** (str) – SNOW Ticket sync string
- **supporteddescriptionfields** (list) – Supported Description fields
- **selected_optional_fields** (list) – Selected optional fields
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
int

**Returns**
Created SNOW connector ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_snow_connector('test',
↪'test' 'pass' 'https://test.com' 'CUSTOM' 'Closed' 'closed' 'Open,In Progress
↪',[],[],client_id=123)
```

**create_snow_service_connector**(*snow_connector_name*, *snow_username*, *snow_password_or_token*, *snow_url*, *ticket_description='false'*, *client_id=None*)

Create a Service Now Service Request type Connector

**Parameters**

- **snow_connector_name** (str) – SNOW Connector Name
- **snow_username** (str) – SNOW username
- **snow_password_or_token** (str) – SNOW API Token/Password
- **snow_url** (str) – SNOW Platform URL

- **ticket_description** (str) – Ticket description
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
int

**Returns**
Created SNOW connector ID

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_snow_service_connector(
↪'test','test','pass','https://test.com',client_id=123)
```

**create_prisma_network_connector**(*conn_name*, *conn_url*, *username*, *password*, *conn_status*, *schedule_freq*, *network_id*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Checkmarx OSA scanning connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **conn_status** (bool) – Whether enabled or disabled
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run, Type : Str, Available values : 1-7
- **day_of_month** (int) – The day of the month the connector should run, Type : Str, Available values : 1-31
- **createAssetsIfZeroVulnFoundInFile** (bool) – Create assets if zero vulnerability found in file. Type : Bool, Available values : True or False
- **maxDaysToRetrieve** (int) – Max days to retrieve scan data. Type : Integer, Available values : 30,60,90,180,365

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_checkmarx_osa_connector(
↪'test', 'https://test.com', 'xxxx', 'xxxx', True, 'DAILY', 123, auto_urba=True,
↪client_id=123, hour_of_day=0)
```

**create_fortify_ondemand_connector**(*conn_name*, *conn_url*, *username*, *password*, *conn_status*, *schedule_freq*, *network_id*, *sdlcstatus*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Checkmarx OSA scanning connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **conn_status** (bool) – Whether enabled or disabled
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run, Type : Str, Available values : 1-7
- **day_of_month** (int) – The day of the month the connector should run, Type : Str, Available values : 1-31
- **createAssetsIfZeroVulnFoundInFile** (bool) – Create assets if zero vulnerability found in file. Type : Bool, Available values : True or False
- **maxDaysToRetrieve** (int) – Max days to retrieve scan data. Type : Integer, Available values : 30,60,90,180,365

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_checkmarx_osa_connector(
↪'test','https://test.com','xxxx','xxxx',True,'DAILY',123,auto_urba=True,
↪client_id=123,hour_of_day=0)
```

**Parameters**
**sdlcstatus** (str) –

**create_sonatype_connector**(*conn_name*, *network_id*, *conn_url*, *username*, *password*, *conn_status*, *schedule_freq*, *taginfopull=True*, *createassetifvulnfound=True*, *nexusapipullapplicationfilter=True*, *nexusapipullstagefilter=True*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Checkmarx OSA scanning connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **conn_status** (bool) – Whether enabled or disabled
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run, Type : Str, Available values : 1-7
- **day_of_month** (int) – The day of the month the connector should run, Type : Str, Available values : 1-31
- **createAssetsIfZeroVulnFoundInFile** (bool) – Create assets if zero vulnerability found in file. Type : Bool, Available values : True or False
- **maxDaysToRetrieve** (int) – Max days to retrieve scan data. Type : Integer, Available values : 30,60,90,180,365

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_checkmarx_osa_connector(
↪'test', 'https://test.com', 'xxxx', 'xxxx', True, 'DAILY', 123, auto_urba=True,
↪client_id=123, hour_of_day=0)
```

**Parameters**

- **taginfopull** (bool) –
- **createassetifvulnfound** (bool) –
- **nexusapipullapplicationfilter** (bool) –
- **nexusapipullstagefilter** (bool) –

**create_checkmarx_osa_connector**(*conn_name*, *conn_url*, *username*, *password*, *conn_status*, *schedule_freq*, *network_id*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Checkmarx OSA scanning connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **conn_status** (bool) – Whether enabled or disabled
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run, Type : Str, Available values : 1-7
- **day_of_month** (int) – The day of the month the connector should run, Type : Str, Available values : 1-31
- **createAssetsIfZeroVulnFoundInFile** (bool) – Create assets if zero vulnerability found in file. Type : Bool, Available values : True or False
- **maxDaysToRetrieve** (int) – Max days to retrieve scan data. Type : Integer, Available values : 30,60,90,180,365

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_checkmarx_osa_connector(
↪'test','https://test.com','xxxx','xxxx',True,'DAILY',123,auto_urba=True,
↪client_id=123,hour_of_day=0)
```

**create_checkmarx_sast_connector**(*conn_name*, *conn_url*, *username*, *password*, *conn_status*, *schedule_freq*, *network_id*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Checkmarx SAST scanning connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **conn_status** (str) – Whether enabled or disabled
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run, Type : Str, Available values : 1-7
- **day_of_month** (int) – The day of the month the connector should run, Type : Str, Available values : 1-31
- **createAssetsIfZeroVulnFoundInFile** (bool) – Create assets if zero vulnerability found in file. Type : Bool, Available values : True or False
- **maxDaysToRetrieve** (int) – Max days to retrieve scan data. Type : Integer, Available values : 30,60,90,180,365

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_checkmarx_sast_connector(
↪'test','https://test.com','xxxx','xxxx',True,'DAILY',123,auto_urba=True,
↪client_id=123,hour_of_day=0)
```

**create_qualys_was**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Qualys Web application connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_qualys_was('test','https:/
↪/test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

**create_qualys_vuln**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Qualys Vulnerability connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_qualyspc('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

---

**create_qualys_asset**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Qualys Asset connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY

- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_qualys_asset('test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**create_nexpose_asset_tag**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Nexpose Asset connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.

- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_nexpose_asset_tag('test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**create_qualys_vmdr**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new qualys vmdr connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_qualys_vmdr('test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

---

**create_nexpose**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Nexpose connector.

**Parameters**

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_nexpose('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

---

**create_teneble**(*conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Create a new Teneble Security Center connector.

**Parameters**

- **conn_name** (str) – The connector name

- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.create_teneble('test','https://
↪test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,hour_of_
↪day=0)
```

`get_connector_detail`(*connector_id*, *client_id=None*)

Get the details associated with a specific connector.

**Parameters**

- **connector_id** (int) – The connector ID.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_connector_detail(123,client_
↪id=123)
```

**update**(*connector_id*, *conn_type*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing connector

**Parameters**

- **connector_id** (int) – Connector ID to update
- **conn_type** (str) – Type of Connector
- **conn_name** (str) – The name for the connector
- **conn_url** (str) – The URL for the connector to communicate with.
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to be used
- **password_or_secret_key** (str) – The password or secret key to be used
- **auto_urba** (bool) – Indicates whether URBA should be automatically run after connector runs.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update(123,'NESSUS','test',
↪'https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**update_expander**(*conn_id*, *conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update expander connector.

**Parameters**

- **conn_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with

- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.[risksenseobject].connectors.update_expander(123,'test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

---

**update_crowdstrike**(*conn_id*, *conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update crowdstrike connector.

**Parameters**

- **conn_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_crowdstrike(123 'test',
↪'https://test.com' 'DAILY' 123 'xxxx' 'xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

---

**update_qualys_vmdr**(*connector_id*, *conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update qualys vm connector.

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_qualys_vmdr(123 'test',
↪'https://test.com' 'DAILY' 123 'xxxx' 'xxxx',auto_urba=True client_id=123
↪hour_of_day=0)
```

**update_nessus_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing Nessus connector

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_nessus_connector(123 'test
↪' 'https://test.com' 123 'DAILY' 'xxxx' 'xxxx',auto_urba=True client_id=123
↪hour_of_day=0)
```

**update_sonarcloud**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing Sonarcloud connector

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_sonarcloud(123,'test',
↪'https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**update_veracode**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing Veracode connector

**Parameters**

- **connector_id** (int) – Connector id

- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_veracode(123, 'test',
↪'https://test.com', 123, 'DAILY', 'xxxx', 'xxxx', auto_urba=True, client_id=123,
↪hour_of_day=0)
```

**update_tenableio**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing Tenable io connector

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY

- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_tenableio(123 'test',
↪'https://test.com' 123 'DAILY' 'xxxx' 'xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**update_teneble**(*connector_id*, *conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing Teneble Security Center connector.

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.

- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_teneble(123 'test',
↪'https://test.com' 'DAILY' 123 'xxxx' 'xxxx',auto_urba=True client_id=123
↪hour_of_day=0)
```

**update_hclappscan**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing hclappscan connector

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.risksenseobject.connectors.update_hclappscan(123,'test',
↪'https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**update_awsinspector**(*connector_id*, *conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing aws inspector connector.

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.risksenseobject.connectors.update_awsinspector(123,'test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**update_burpsuite**(*connector_id*, *conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *apikey*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing burpsuite connector.

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication
- **apikey** (str) – The apikey to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_burpsuite(123,'test',
↪'https://test.com','DAILY',123,'xxxx','xxxx',auto_urba=True,client_id=123
↪hour_of_day=0)
```

**update_nexpose**(*connector_id*, *conn_name*, *conn_url*, *schedule_freq*, *network_id*, *username*, *password*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing Nexpose connector.

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **username** (str) – The username to use for connector authentication

- **password** (str) – The password to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.frisksenseobject.connectors.update_nexpose(123, 'test',
↪'https://test.com', 'DAILY', 123, 'xxxx', 'xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**update_qualys_vm_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing QUALYS VM connector

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.

- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_qualys_vm_connector(123,
↪'test','https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_
↪id=123,hour_of_day=0)
```

**update_qualys_pc_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing QUALYS PC connector

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_qualys_pc_connector(123,
↪'test','https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_
↪id=123,hour_of_day=0)
```

**update_nexpose_asset**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing QUALYS PC connector

**Parameters**

- **connector_id** (int) – Connector id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_nexpose_asset(123,'test',
↪'https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_id=123,
↪hour_of_day=0)
```

**search_query_parameter**(*body*, *client_id=None*)

Get option drop down fields based on search query parameters

**Parameters**

- **body** (dict) – The body that contains connector information
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned

---

**Note:** Intercept this API endpoint /client/{clientId}/connector/field/option in UI to better understand the body that need to be sent using this function. Then, use this function in your automation.

---

**update_jira_connector**(*connector_id*, *jira_connector_name*, *username*, *password_or_api_token*, *jira_url*, *project_name*, *project_key*, *issue_type_name*, *issue_type_key*, *tag_type_name*, *closed_status_key*, *closed_status_value*, *ticket_sync_string*, *client_id=None*)

Updates a JIRA Connector

**Parameters**

- **connector_id** (int) – The Connector Id
- **jira_connector_name** (str) – JIRA Connector Name
- **username** (str) – JIRA username
- **password_or_api_token** (str) – JIRA API Token/Password
- **jira_url** (str) – JIRA Platform URL
- **project_name** (str) – JIRA Project Name
- **project_key** (str) – JIRA Project Key
- **issue_type_name** (str) – JIRA Issue Type Name
- **issue_type_key** (str) – JIRA Issue Type Key
- **tag_type_name** (str) – JIRA Issue Type Name
- **closed_status_key** (str) – JIRA Closed status Key
- **closed_status_value** (str) – JIRA Closed status name
- **ticket_sync_string** (str) – JIRA Ticket sync string
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
int

**Returns**
Created JIRA connector ID

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_jira_connector(123,'test',
↪'xxxx','xxxx','https://test.com','test project','TP','bug','bug','CUSTOM',
↪'closed','Closed','Open,In Progress',client_id=123)
```

---

**update_snow_connector**(*connector_id*, *snow_connector_name*, *snow_username*, *snow_password_or_token*, *snow_url*, *tag_type_name*, *closed_status_value*, *closed_status_key*, *ticket_sync_string*, *client_id=None*)

Update a Service Now Incident type Connector

**Parameters**

- **connector_id** (int) – The Connector Id
- **snow_connector_name** (str) – SNOW Connector Name
- **snow_username** (str) – SNOW username
- **snow_password_or_token** (str) – SNOW API Token/Password
- **snow_url** (str) – SNOW Platform URL
- **tag_type_name** (str) – SNOW Issue Type Name
- **closed_status_value** (str) – SNOW Closed status name
- **closed_status_key** (str) – SNOW Closed status Key
- **ticket_sync_string** (str) – SNOW Ticket sync string
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
int

**Returns**
Created SNOW connector ID

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_snow_connector(123,'test',
↪'xxxx','xxxx','https://test.com','CUSTOM','Closed','closed','Open,In Progress
↪',client_id=123)
```

---

**update_snow_customtableconfig**(*connector_id*, *conn_name*, *conn_url*, *username*, *password*, *tablename*, *statusfield*, *ticketidfield*, *enabletagremoval=False*, *enableuploadattachment=True*, *client_id=None*)

Update an existing snow custom table configuration connector.

**Parameters**

- **connector_id** (int) – The Connector Id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication

- **tablename** (str) – Name of the table
- **statusfield** (str) – status field
- **ticketidfield** (str) – Ticket id field
- **enabletagremoval** (bool) – Enable tag removal switch
- **enableuploadattachment** (bool) – Enable upload attachment switch
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_snow_
↪customtableconfig(123,'test','https://test.com','xxxx','xxxx','test','new',
↪'123',enabletagremoval=True,client_id=123,enableuploadattachment=True)
```

---

**update_snow_service_connector**(*snow_connectorid*, *snow_connector_name*, *snow_username*, *snow_password_or_token*, *snow_url*, *ticket_description='false'*, *client_id=None*)

Update existing Service Now Service Request type Connector

**Parameters**

- **snow_connectorid** (int) – The Connector Id
- **snow_connector_name** (str) – SNOW Connector Name
- **snow_username** (str) – SNOW username
- **snow_password_or_token** (str) – SNOW API Token/Password
- **snow_url** (str) – SNOW Platform URL
- **ticket_description** (str) – To provide ticket description or not,'true' to provide,'false' to not provide
- **client_id** (typing.Optional[int]) – RS Client ID

**Return type**
int

**Returns**
Created SNOW Service request connector ID

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_snow_service_
↪connector(123,'test','xxxx','xxxx','https://test.com',client_id=123)
```

---

**update_cherwell_incident_connector**(*cw_id*, *cw_name*, *cw_username*, *cw_password*, *clientid_key*, *cw_url*, *autourba=False*, *client_id=None*)

Update cherwell incident type connector

**Parameters**

- **cw_id** (int) – The Connector Id
- **cw_name** (str) – Cherwell connector name.
- **cw_username** (str) – Cherwell connector username.
- **cw_password** (str) – Cherwell connector password.
- **clientid_key** (str) – Cherwell connector client id key.
- **cw_url** (str) – Cherwell connector url
- **autourba** (bool) – Switch to enable auto urba
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_cherwell_incident_
↪connector(123,'test','xxxx','xxxx','xxxx','https://test.com',auto_urba=True,
↪client_id=123)
```

---

**update_cherwell_problem_connector**(*cw_id*, *cw_name*, *cw_username*, *cw_password*, *clientid_key*, *cw_url*, *autourba=False*, *client_id=None*)

Update cherwell problem type connector

**Parameters**

- **cw_id** (int) – The Connector Id
- **cw_name** (str) – Cherwell connector name.
- **cw_username** (str) – Cherwell connector username.
- **cw_password** (str) – Cherwell connector password.
- **clientid_key** (str) – Cherwell connector client id key.
- **cw_url** (str) – Cherwell connector url
- **autourba** (bool) – Switch to enable auto urba
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_cherwell_problem_
↪connector(123,'test','xxxx','xxxx','xxxx','https://test.com',auto_urba=True,
↪client_id=123)
```

**update_cherwell_makerequest_connector**(*cw_id*, *cw_name*, *cw_username*, *cw_password*, *clientid_key*, *cw_url*, *autourba=False*, *client_id=None*)

Updates an existing cherwell change request type connector

**Parameters**

- **cw_id** (int) – The Connector Id
- **cw_name** (str) – Cherwell connector name.
- **cw_username** (str) – Cherwell connector username.
- **cw_password** (str) – Cherwell connector password.
- **clientid_key** (int) – Cherwell connector client id key.
- **cw_url** (str) – Cherwell connector url
- **autourba** (bool) – Switch to enable auto urba
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_cherwell_makerequest_
↪connector(123,'test','xxxx','xxxx','xxxx','https://test.com',auto_urba=True,
↪client_id=123)
```

**update_checkmarx_osa_connector**(*connector_id*, *conn_name*, *conn_url*, *username*, *password*, *conn_status*, *schedule_freq*, *network_id*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update a new Checkmarx OSA scanning connector.

**Parameters**

- **connector_id** (int) – The connector ID.
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **conn_status** (bool) – Whether enabled or disabled

- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run, Type : Str, Available values : 1-7
- **day_of_month** (int) – The day of the month the connector should run, Type : Str, Available values : 1-31
- **createAssetsIfZeroVulnFoundInFile** (bool) – Create assets if zero vulnerability found in file. Type : Bool, Available values : True or False
- **maxDaysToRetrieve** (int) – Max days to retrieve scan data. Type : Integer, Available values : 30,60,90,180,365

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_checkmarx_osa_
↪connector(123,'test','https://test.com','xxxx','xxxx',True,'DAILY',123,auto_
↪urba=True,client_id=123,hour_of_day=0)
```

**update_checkmarx_sast_connector**(*connector_id*, *conn_name*, *conn_url*, *username*, *password*, *conn_status*, *schedule_freq*, *network_id*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update a new Checkmarx SAST scanning connector.

**Parameters**

- **connector_id** (int) – The connector ID.
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **username** (str) – The username to use for connector authentication
- **password** (str) – The password to use for connector authentication
- **conn_status** (bool) – Whether enabled or disabled

- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **network_id** (int) – The network ID
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run, Type : Str, Available values : 1-7
- **day_of_month** (int) – The day of the month the connector should run, Type : Str, Available values : 1-31
- **createAssetsIfZeroVulnFoundInFile** (bool) – Create assets if zero vulnerability found in file. Type : Bool, Available values : True or False
- **maxDaysToRetrieve** (int) – Max days to retrieve scan data. Type : Integer, Available values : 30,60,90,180,365

**Return type**
int

**Returns**
The connector ID from the platform is returned.

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_checkmarx_sast_
↪connector(123,'test','https://test.com','xxxx','xxxx',True,'DAILY',123,auto_
↪urba=True,client_id=123,hour_of_day=0)
```

**update_qualys_was_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing QUALYS was connector

**Parameters**

- **connector_id** (int) – The connector Id
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication

- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** – Automatically run URBA after connector runs?
- **client_id** – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.[risksenseobject].connectors.update_qualys_was_connector(123,
↪'test','https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_
↪id=123,hour_of_day=0)
```

**update_qualys_vuln_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing QUALYS VULN connector

**Parameters**

- **connector_id** (int) – Connector ID to update
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31
- **reportnameprefix** (str) – Report Name Prefix

**Return type**
int

**Returns**
The connector ID from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_qualys_vuln_connector(123,
↪'test','https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_
↪id=123,hour_of_day=0)
```

**update_qualys_asset_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing QUALYS ASSET connector

**Parameters**

- **connector_id** (int) – Connector ID to update
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.[risksenseobject].connectors.update_qualys_asset_
↪connector(123,'test','https://test.com',123,'DAILY','xxxx','xxxx',auto_
↪urba=True,client_id=123,hour_of_day=0)
```

---

**update_nexpose_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing NEXPOSE connector

**Parameters**

- **connector_id** (int) – Connector ID to update
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_nexpose_connector(123
↪'test','https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_
↪id=123,hour_of_day=0)
```

---

**update_teneble_connector**(*connector_id*, *conn_name*, *conn_url*, *network_id*, *schedule_freq*, *username_or_access_key*, *password_or_secret_key*, *auto_urba=True*, *client_id=None*, ***kwargs*)

Update an existing TENEBLE SECURITY CENTER connector

**Parameters**

- **connector_id** (int) – Connector ID to update
- **conn_name** (str) – The connector name
- **conn_url** (str) – The URL for the connector to communicate with
- **network_id** (int) – The network ID
- **schedule_freq** (str) – The frequency for the connector to run. Options: Connectors.ScheduleFreq.DAILY, Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **username_or_access_key** (str) – The username or access key to use for connector authentication
- **password_or_secret_key** (str) – The password or secret key to use for connector authentication
- **auto_urba** (bool) – Automatically run URBA after connector runs?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Keyword Arguments**

- **ssl_cert** (str) – Optional SSL certificate.
- **hour_of_day** (int) – The time the connector should run. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Integer. 1-31

**Return type**
int

**Returns**
The connector ID from the platform is returned

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_teneble_connector(123
↪'test','https://test.com',123,'DAILY','xxxx','xxxx',auto_urba=True,client_
↪id=123,hour_of_day=0)
```

---

**delete**(*connector_id*, *delete_tag=True*, *client_id=None*)

Delete a connector.

**Parameters**

- **connector_id** (int) – The connector ID
- **delete_tag** (bool) – Force delete tag associated with connector?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Returns**
Indicator reflecting whether or not the operation was successful.

**Return type**
bool

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.delete(123 delete_tag=True
↪client_id=123)
```

---

**get_jobs**(*connector_id*, *page_num=0*, *page_size=150*, *csvdump=False*, *client_id=None*)

Get the jobs associated with a connector.

**Parameters**

- **connector_id** (int) – The connector ID
- **page_num** (int) – The page number of results to be returned
- **page_size** (int) – The number of results to return per page
- **csvdump** (bool) – Whether to dump the assessment history in a csv, true to dump and false to not dump
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
The JSON response from the platform is returned.

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_jobs(123 0 100 client_id=123)
```

---

**Note:** You can also dump the data of the connector jobs in a csv file. Just make csvdump as True:

```
>>> self.{risksenseobject}.connectors.get_jobs(123 0 100 client_id=123
↪csvdump=True)
```

---

**get_logs**(*connector_id*, *page_num=0*, *page_size=150*, *csvdump=False*, *client_id=None*)

Get the jobs associated with a connector.

**Parameters**

- **connector_id** (int) – The connector ID

- **page_num** (int) – The page number of results to be returned
- **page_size** (int) – The number of results to return per page
- **csvdump** (bool) – Whether to dump the assessment history in a csv, true to dump and false to not dump
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
The JSON response from the platform is returned

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.get_logs(123 0 100 client_id=123)
```

**Note:** You can also dump the data of the connector jobs in a csv file. Just make csvdump as True:

```
>>>  self.{risksenseobject}.connectors.get_logs(123 0 100 client_id=123
↪csvdump=True)
```

**update_schedule**(*connector_id*, *schedule_freq*, *enabled*, *client_id=None*, ***kwargs*)

Update the schedule of an existing Connector.

**Parameters**

- **connector_id** (int) – Connector ID
- **schedule_freq** (str) – The frequency for the connector to run. Connectors.ScheduleFreq.DAILY,Connectors.ScheduleFreq.WEEKLY, Connectors.ScheduleFreq.MONTHLY
- **enabled** (bool) – Enable connector?
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Keyword Arguments**

- **hour_of_day** (int) – The time the connector should run. Req. for DAILY, WEEKLY, and MONTHLY. Integer. 0-23.
- **day_of_week** (int) – The day of the week the connector should run. Req. for WEEKLY. Integer. 1-7
- **day_of_month** (int) – The day of the month the connector should run. Req. for MONTHLY. Integer. 1-31

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.update_schedule(123,'DAILY',True,
↪client_id=123,hour_of_day=0)
```

**itsm_get_fields_for_ticket_type**(*ticket_type*, *connector_url*, *connector_name*, *api_key*, *username='admin'*, *client_id=None*)

Get fields available for ticket type of a connector

**Parameters**

- **ticket_type** (str) – Ticket type
- **connector_url** (str) – ITSM Connector URL
- **connector_name** (str) – ITSM Connector Name
- **api_key** (str) – ITSM API Key
- **username** (str) – ITSM Username, defaults to 'admin'
- **client_id** (typing.Optional[int]) – RS Client Id, defaults to None

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.itsm_get_fields_for_ticket_type(
↪'incident','https://test.com','test','xxxx',client_id=123)
```

**ivanti_itsm_fetch_customers**(*itsm_url*, *itsm_api_key*, *client_id=None*)

Fetch ITSM available customers

**Parameters**

- **itsm_url** (str) – ITSM Connector URL
- **itsm_api_key** (str) – ITSM API Key
- **client_id** (typing.Optional[int]) – RS Client Id, defaults to None

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.ivanti_itsm_fetch_customers(
↪'https://test.com','xxxx',client_id=123)
```

**ivanti_itsm_fetch_fieldValue_wrt_dependentField**(*ticket_type*, *itsm_url*, *itsm_api_key*, *current_field*, *dependent_field*, *dependent_field_value*, *client_id=None*)

Fetch value for a ITSM connector field with respect to dependent field

**Parameters**

- **ticket_type** (str) – Ticket type
- **itsm_url** (str) – ITSM Connector URL
- **itsm_api_key** (str) – ITSM API Key
- **current_field** (str) – Field that should be queried for available options
- **dependent_field** (str) – Dependent field
- **dependent_field_value** (str) – Dependent field value
- **client_id** (typing.Optional[int]) – RS Client Id, defaults to None

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.ivanti_itsm_fetch_fieldValue_wrt_
↪dependentField('incident','https://test.com','xxxx','category','status','test
↪',client_id=123)
```

---

**ivanti_itsm_fetch_validation**(*body*, *ticket_type*, *client_id=None*)

ITSM Connector field form validation

**Parameters**

- **body** (str) – Form validation request body
- **ticket_type** (str) – Ticket type
- **client_id** (typing.Optional[int]) – RS Client Id, defaults to None

**Return type**
dict

**Returns**
Jsonified response

---

**Note:** Intercept this API endpoint `/client/{clientId}/connector/ivanti/formValidation/ticketType/{ticket_type}` in UI to better understand the body that need to be sent using this function. Then, use this function in your automation.

---

**ivanti_itsm_fetch_releaseLink**(*itsm_url*, *itsm_api_key*, *client_id=None*)

Fetch available options for Release Link field

**Parameters**

- **itsm_url** (str) – ITSM Connector URL

- **itsm_api_key** (str) – ITSM API Key
- **client_id** (typing.Optional[int]) – RS Client Id, defaults to None

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.ivanti_itsm_fetch_releaseLink(
↪'https://test.com','xxxx',client_id=123)
```

**ivanti_itsm_fetch_requestorLink**(*itsm_url*, *itsm_api_key*, *client_id=None*)

Fetch available options for Requestor Link field

**Parameters**

- **itsm_url** (str) – ITSM Connector URL
- **itsm_api_key** (str) – ITSM API Key
- **client_id** (typing.Optional[int]) – RS Client Id, defaults to None

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.connectors.ivanti_itsm_fetch_requestorLink(
↪'https://test.com','xxxx',client_id=123)
```

**create_ivanti_itsm_connector**(*body*, *client_id=None*)

Create ITSM Connector

**Parameters**

- **body** (dict) – Ticket request body
- **client_id** (typing.Optional[int]) – RS Client Id, defaults to None

**Return type**
dict

**Returns**
Jsonified response

**Note:** Intercept this API endpoint `/client/{clientId}/connector` in UI for ITSM connector to better understand the body that need to be sent using this function. Then, use this function in your automation.

### 1.2.28 Tickets (risksense_api.__subject.__ticket.__ticket)

Ticket module defined for different ticket related api endpoints.

**class** risksense_api.__subject.__ticket.__ticket.**Ticket**(*profile*)

Bases: Subject

Class for ticket function definitions.

**Parameters**
**profile** (object) – Profile Object

To utlise ticket function:

**Usage:**
self.{risksenseobjectname}.ticket.{function}

---

**Examples**

To get connector fields for a ticket using *getconnectorfields()* function

```
>>> self.{risksenseobjectname}.ticket.getconnectorfields(args)
```

---

**__init__**(*profile*)

Initialization of Ticket object.

**Parameters**
**profile** (object) – Profile Object

**getconnectorfields**(*connector_id*, *client_id=None*)

Get connector fields present in ticket form

**Parameters**

- **connector_id** (int) – Connector Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.getconnectorfields(123, client_id=123)
```

---

**gettemplateid**(*connector_id*, *client_id=None*)

Get available templates

**Parameters**

- **connector_id** (int) – Connector Id

- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.gettemplateid(123 client_id=123)
```

---

**getfieldsfromtemplateid**(*connector_id*, *template_id*, *client_id=None*)

Get fields available for a particular template

**Parameters**

- **connector_id** (int) – Connector Id
- **template_id** (int) – Template Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.getfieldsfromtemplateid(123 123
↪client_id=123)
```

---

**getcatalogitemfield**(*connector_id*, *client_id=None*)

Get fields available for a particular catalog item

**Parameters**

- **connector_id** (int) – Connector Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.getcatalogitemfield(123 client_
↪id=123)
```

---

**getissuetypefield**(*connector_id*, *client_id=None*)

Get available issue types

**Parameters**

- **connector_id** (int) – Connector Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.getissuetypefield(123,client_id=123)
```

---

**getticketinfo**(*ticket_id*, *client_id=None*)

Get info about a ticket

**Parameters**

- **ticket_id** (str) – Ticket Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.getticketinfo('TP-123',client_id=123)
```

---

**deleteticket**(*ticket_uuid*, *client_id=None*)

Delete ticket

**Parameters**

- **ticket_uuid** (str) – Ticket UUID
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
bool

**Returns**
Deletion status

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.getconnectorfields('123-456',client_
↪id=123)
```

**create_ticket**(*tag_id*, *body*, *client_id=None*)

Create a ticket

**Parameters**

- **tag_id** (int) – Tag Id
- **body** (dict) – API request ticket body
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
Jsonified response

**Note:** Intercept this API endpoint `/client/{clientId}/ticket/{tag_id}` in UI to better understand the body that need to be sent using this function. Then, use this function in your automation.

**ivanti_itsm_fetch_ticketField_values**(*connector_id*, *client_id=None*)

Fetch ticket field values for Ivanti ITSM

**Parameters**

- **connector_id** (int) – Connector Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.ivanti_itsm_fetch_ticketField_
↪values(123,client_id=123)
```

**ivanti_itsm_retrieve_ticketFields**(*connector_id*, *ticket_type*, *client_id=None*)

Retrieve ticket fields for Ivanti ITSM

**Parameters**

- **connector_id** (int) – Connector Id
- **ticket_type** (str) – Type of the ticket to be created.
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.ivanti_itsm_retrieve_
↪ticketFields(123,'incident',client_id=123)
```

---

**ivanti_itsm_fetch_customers**(*connector_id*, *client_id=None*)

Fetch available customers for Ivanti ITSM

**Parameters**

- **connector_id** (int) – Connector Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.ivanti_itsm_fetch_customers(123
↪client_id=123)
```

---

**ivanti_itsm_fetch_fieldValue_wrt_dependentField**(*ticket_type*, *connector_id*, *current_field*, *dependent_field*, *dependent_field_value*, *client_id=None*)

Fetch available options for a current Ivanti ITSM field with respect to its dependent Ivanti ITSM field

**Parameters**

- **ticket_type** (str) – Ticket type
- **connector_id** (int) – Connector Id
- **current_field** (str) – Field name key of the field to be queried for
- **dependent_field** (str) – Dependent field name key
- **dependent_field_value** (str) – Value of the dependent field
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.ivanti_itsm_fetch_fieldValue_wrt_
↪dependentField('incident',123,'category','status','test',client_id=123)
```

**ivanti_itsm_fetch_validation**(*body*, *client_id=None*)

Form validation for ivanti ITSM

**Parameters**

- **body** (dict) – API request ticket body
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID

**Return type**
dict

**Returns**
Jsonified response

**Note:** Intercept this API endpoint /client/{clientId}/connector/{connectorId}/ivanti/formValidation in UI to better understand the body that need to be sent using this function. Then, use this function in your automation.

**ivanti_itsm_fetch_releaseLink**(*connector_id*, *client_id=None*)

Fetch available release links for Ivanti ITSM

**Parameters**

- **connector_id** (int) – Connector Id
- **client_id** (typing.Optional[int]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

**Examples**

```
>>> apiobj = self.{risksenseobject}.ticket.ivanti_itsm_fetch_releaseLink(123,
↪client_id=123)
```

**ivanti_itsm_fetch_requestorLink**(*connector_id*, *client_id=None*)

Fetch available requestor links for Ivanti ITSM

**Parameters**

- **connector_id** (int) – Connector Id
- **client_id** (typing.Optional[list]) – Client ID. If an ID isn't passed, will use the profile's default Client ID.

**Return type**
dict

**Returns**
Jsonified response

---

**Examples**

```
>>> apiobj = self.[risksenseobject].ticket.ivanti_itsm_fetch_requestorLink(123
↪client_id=123)
```

---

CHAPTER

TWO

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

r

# INDEX

## Symbols



## A

## D

## E



## F



## G

## H

## I



## J



## L

## M



## N

## S

## T



## U

## V



## W



## X