TRAW Documentation

Levi Noecker

Feb 13, 2019

Contents:

TRAW, the TestRail Api Wrapper, aims to be a feature complete python interface to TestRail's RESTful API.

CHAPTER 1

Contents

1.1 Quick Reference

```
from datetime import datetime as dt
import random

import traw

client = traw.Client(username='username', password='passwrod', url='url)

project = client.project(15)  # Get Project with Project ID of 15

new_run = client.run()
new_run.name = "My new run name"
new_run.description = "My new run description"
new_run.include_all = True
new_run.project = project

run = client.add(new_run)  # Run is added to TestRail

for test in client.tests(run):  # Get all tests for run
    begin = dt.now()

    # Do actual testing here, but lets pick a random status
    status_str = random.choice(['passed', 'failed', 'retest'])

    elapsed = dt.now() - begin

    result = client.result()
    result.test = test
    result.status = client.status(status_str)
    result.comment = "Setting {0} to {1}".format(test.title, result.status.label)
    result.elapsed = elapsed
```

(continues on next page)

(continued from previous page)

```
    # Add the result to TestRail
    client.add(result)

# Everything complete, close the run
client.close(run)

# Fín
```

1.2 Credentials

The TRAW Client can pull in credentials in three ways:

- Passing parameters to __init__ during traw.Client instantiation

```
client = traw.Client(username='user@email.com', password='password', url='url')
# client = traw.Client(username='user@email.com', user_api_key='userapikey', url=
↪'url')
```

- Setting environment variables

```
$ export TRAW_USERNAME="user@email.com"
$ export TRAW_PASSWORD="userapikey"
$ # export TRAW_USER_API_KEY="userapikey"  # (Optional) - in place of TRAW_
↪PASSWORD
$ export TRAW_URL="https://example.testrail.net"
```

- Writing them to a configuration file in the user's home directory

```
$ cat ~/.traw_config
[TRAW]
username = <username>
password = <password>
# user_api_key = <user_api_key>  # (Optional) - in place of password
url = <url>
```

You can create multiple clients to access different TestRail installations:

```
client1 = traw.Client(username='user1@email.com', password='password', url='https://
↪example.testrail.net')
client2 = traw.Client(username='user2@email.com', password='password', url='https://
↪your.domain.com')
```

1.3 Object Manipulation

TRAW uses a consistent pattern for creating new TestRail objects and adding them to TestRail:

- Call the relevant client method without any parameters, and a new/empty object is returned:

```
new_run = client.run()
new_result = client.result()
new_section = client.section()
new_milestone = client.milestone()
# etc
```

- Configure the new object. Note most addable objects require at least one reference object in order for them to be added to TestRail. For instance, run objects require a reference to a project, result objects require a reference to a test, and sections objects require a reference to a project AND a suite if the project is not in single-suite mode:

```
new_run.name = "Run Name"
new_run.project = client.project(15)  # Project with Project ID 15

new_result.comment = "Result added by TRAW"
new_result.test = client.tests(123)  # Test with Test ID of 123
new_result.status = client.status('passed')  # Status with Status Label of 'passed
↪'

new_section.name = "Suite Name"
new_section.project = client.project(15)  # Project with Project ID 15, with␣
↪suite-mode of 2
new_section.suite = client.suite(456)  # Suite with Suite ID 456
```

- At this point the objects only exist locally, and have not been added to TestRail. To do so, call `client.add()` with the new object. TRAW will add the new object to TestRail, and upon success the TestRail API will return a new object:

```
run = client.add(new_run)
result = client.add(result)
section = client.add(section)
```

- The returned objects will now have additional information set. Properties that have not yet been specified will be set to None:

```
print("Run ID is: {0}".format(run.id))                                    # "Run ID is:␣
↪12333"
print("Run Name is: '{0}'".format(run.name))                              # "Run Name is:
↪'Run Name'"
print("Run Created By user: '{0}'".format(run.created_by.name))           # "Run Created␣
↪By user: 'Automation User'"
print("Run Created On: '{0}'".format(run.created_on))                     # "Run Created␣
↪On: '2016-08-19 13:00:29'"
print("Run Milestone: '{0}'".format(run.milestone))                       # "Run␣
↪Milestone: 'None'"
```

- Objects that support updating (runs, suites, milestones, etc) can be updated locally, and then the updates can be sent to TestRail:

```
run.name = run.name + " - Updated by TRAW"
run.milestone = client.milestone(789)

updated_run = client.update(run)

print("Run Name is: '{0}'".format(updated_run.name))                      # "Run Name is:
↪'Run Name - Updated by TRAW'"
print("Run Milestone: '{0}'".format(updated_run.milestone))               # "Run␣
↪Milestone: 'Widget Testing 90% Complete'"
print("Run Complete: '{0}'".format(updated_run.is_completed))             # "Run␣
↪Completed: 'False'"
```

- Objects that can be closed (runs, plans) can be closed through the TRAW Client:

```
closed_run = client.close(run)

print("Run Complete: '{0}'".format(closed_run.is_completed))      # "Run␣
↪Completed: 'True'"
```

- Objects that can be deleted (runs, plans, cases, etc) can be deleted through the TRAW Client. Note that no object is returned after calling `client.delete()`. Also note that some things (runs, plans) can either be closed or deleted, but not both, while other things (projects) can be deleted after they have been closed (assuming your user has admin privileges):

```
client.delete(run)
```

See also:

- TRAW Examples: TRAW examples scripts

CHAPTER 2

Indices and tables

- genindex
- modindex
- search