Camayoc

Jun 06, 2023

Contents:

CHAPTER 1

Documentation for developers

This is Camayoc development documentation. It is mostly generated manually by actual human beings. While still not a gospel, it allows to focus on things that would be easy to miss in automatically generated documentation and gives space to discuss things that go beyond the scope of package API (i.e. "Why?", instead of only "What?" and "How?").

1.1 UI tests framework

Caution: This documentation focuses on UI testing framework that uses Playwright. Older framework, built on top of Selenium, is briefly documented in project README file.

1.1.1 Preparing Playwright environment

Before running Playwright-based tests, Playwright environment must be prepared. In Camayoc virtual environment, run:

```
playwright install
```

Playwright on Linux (and other operating systems?) requires certain system-level dependencies to be met. You can install them with command:

```
playwright install-deps
```

Add `--dry-run` to see what Playwright wants to install.

You can verify that Playwright is set up correctly by running:

```
playwright cr 'https://redhat.com'  # Opens Chromium
playwright ff 'https://redhat.com'  # Opens Firefox
playwright wk 'https://redhat.com'  # Opens WebKit
```

Note: Running Firefox on Fedora

Offically, Playwright supports only Ubuntu LTS releases. `playwright ff` might error out on Fedora system. The fix is to use system-wide libstdc++:

```
rm ~/.cache/ms-playwright/firefox-1323/firefox/libstdc++.so.6
ln -s /usr/lib64/libstdc++.so.6 ~/.cache/ms-playwright/firefox-1323/firefox/libstdc++.
↪so.6
```

You might have different version of Firefox, so change directory name above accordingly.

1.1.2 Running existing UI tests

Use pytest, just like for all the other tests:

```
pytest camayoc/tests/qpc/ui/test_endtoend.py::test_demo_endtoend -s -v
```

By default, Playwright will run tests in **headless** mode, i.e. you won't see anything. Add `--headed` to see what Playwright is doing.

Playwright tries to run tests in all three supported engines (Chromium, Firefox, WebKit). Add `--browser chromium` to run tests using specific browser.

See pytest-playwright documentation for list of all supported command-line options. Playwright also maintains a page with debugging tips.

1.1.3 Framework building blocks

Let's discuss the building blocks of the framework. Some parts of this section might be clearer after reading "Design decisions" section below, but the same is true for that section. "What?" and "Why?" are kind of intertwined.

Client

Similar to API and CLI interfaces, there's `camayoc.ui.Client` class representing UI client. This is your main entry point to interact with Quipucords web interface. It wraps Playwright's `Page` class (which represents single tab in the browser) and coordinates work of few other classes. `ui_client` pytest fixture returns `Client` instance. Each test will get new instance with completely new browser window. Starting browser in Playwright is fast, so there's little reason to re-use browser between tests.

Page models

In accordance with Page Object Model design pattern, each page is represented by single class.

Page objects aim to make tests more maintainable by limiting the number of places where changes must be made. For example, most tests would need to log in at the beginning. If you use Playwright directly, your test could look like that:

```
client.driver.fill("#username", username)
client.driver.fill("#password", password)
client.driver.click("#login")
```

Now imagine that "Log in" button id attribute changed from `login` to `submit`. Suddenly you need to change element locator in all tests.

With page objects, the code above is moved to `login` method of `LoginPage` object, and tests will look like that:

```
LoginPage().login(username, password)
```

When element on page changes, there's limited number of places that have to be changed - ideally, there will be only one such place.

Page objects should provide methods that define actions that user can make on specific page. In Camayoc, we err on the side of higher level actions. So `LoginPage` will have single `login()` method, instead of triplet of `fill_username()`, `fill_password()` and `submit_form()` methods. The idea is that after performing an action, browser should be left in a state where it's reasonable to take another action. Another idea is that actions should make sense on their own and should not require other actions to complete before they can be invoked. (Note that this applies to interface itself and not business logic of Quipucords. It doesn't make much sense to go to Scans before creating Source, but UI allows user to do that, so framework should expose such action.)

Chains of related actions should be wrapped in "page service" methods. These allow to write shorter tests, which is especially important for code that sets up the stage for actual verification. For example, creating new source requires filling two forms in three-step wizard. `SourcePage` may expose `create_source` "service", which essentially does:

```
SourcePage()
  .open_add_source_wizard()
  .fill(wizard_step1_data)
  .next_step()
  .fill(wizard_step2_data)
  .next_step()
  .close_wizard()
```

Page components

Some components are common to multiple pages, like vertical menu or logout button. In Camayoc, we separate these into mixin classes and use multiple inheritance to compose pages with shared capabilities. Multiple inheritance is preferred over composition, because it allows to skip method-forwarding code. Compare:

Listing 1: Composition

```
class VerticalNavigationComponent(AbstractPage):
    def navigate_to(self, destination):
        ...

class LogoutComponent(AbstractPage):
    def logout(self):
        ...

class SourcePage(AbstractPage):
    def __init__(self):
        self.vertical_nav_component = VerticalNavigationComponent()
        self.logout_component = LogoutComponent()

    def navigate_to(self, destination):
        self.vertical_nav_component.navigate_to(destination)
```

(continues on next page)

(continued from previous page)

```
    def logout(self):
        self.logout_component.logout()
```

Listing 2: Multiple inheritance

```
class VerticalNavigationComponent:
    def navigate_to(self, destination):
        ...

class LogoutComponent:
    def logout(self):
        ...

class SourcePage(LogoutComponent, VerticalNavigationComponent, AbstractPage):
    # mro takes care of calling VerticalNavigationComponent.navigate_to()
    # and LogoutComponent.logout()
    ...
```

Forms

Forms handling is highly inspired by Django / Django REST Framework.

Page object that contains a form should inherit `camayoc.ui.models.components.form.Form` component. They should also define class `FormDefinition` as class property. `FormDefinition` should contain properties for each form field. Property name should match input data object attribute name (see below), and property value should be an instance of class that inherit from `Field` class. `Field` class instantiation takes two arguments: selector that finds this specific field on this specific page, and optional function that may be used to transform input data into something that Playwright can understand.

Complete basic example:

```
from .abstract_page import AbstractPage
from ..components.form import Form
from ..fields import InputField


class SomeForm(Form, AbstractPage):
     class FormDefinition:
         user_id = InputField("input#id", lambda i: str(i))
```

Types

All page methods should take up to one additional argument. This argument should be either an enum, or an instance of special data-input class (DTO). We use `enum` module from Python standard library for enums, and attrs for data-input classes.

There are three main reasons for that. First, to limit the space of values that methods need to work with. We don't want to obfuscate page object methods with input validation logic. Page object methods should be able to assume that their arguments have certain properties, and type hints are the easiest way of achieving that.

Second, to catch data input mistakes early. When data is transferred with dictionaries, typos in key names and missing required keys are discovered only at runtime. This is frustrating for test developers, especially if mistake was made relatively late in the test and they have to wait a long time before fix may be verified. With strongly-typed input, your editor should notify you that method input is of wrong type, and mistakes can be corrected much earlier.

Third, there are libraries to generate objects with random data.

Example below shows how to use enum as method input, how to create simple DTO and how to create complex DTO.

```
from camayoc.ui.types import AddCredentialDTO
from camayoc.ui.types import LoginFormDTO
from camayoc.ui.types import SSHNetworkCredentialFormDTO
from camayoc.ui.enums import CredentialTypes
from camayoc.ui.enums import MainMenuPages
from camayoc.ui.enums import NetworkCredentialBecomeMethods
from camayoc.utils import uuid4

SourcesMainPage().navigate_to(MainMenuPages.CREDENTIALS)

login_data = LoginFormDTO(username='admin', password='admin')

credential_data = AddCredentialDTO(
    credential_type=CredentialTypes.NETWORK,
    credential_form_dto=SSHNetworkCredentialFormDTO(
        credential_name="my credential name " + uuid4(),
        username="username" + uuid4(),
        ssh_key_file="/root/.bashrc",
        passphrase="supersecretpassword" + uuid4(),
        become_method=NetworkCredentialBecomeMethods.PFEXEC,
        become_user="systemusername" + uuid4(),
        become_password="systemsecretpassword" + uuid4(),
    ),
)
```

Creating random test data

Camayoc uses factory_boy to generate DTOs with random data. factory_boy uses Faker under the hood.

factory_boy provides highly declarative syntax and allows for related fields, i.e. cases where value of one field is constrained on value of another field. It's easy to generate complete objects, and it's easy to specify values of selected fields.

Example below shows how to create simple DTO, how to specify value for single field and how to create complex DTO while setting value in related object.

```
from camayoc.ui.data_factories import LoginFormDTOFactory
from camayoc.ui.data_factories import AddCredentialDTOFactory
from camayoc.ui.data_factories import AddSourceDTOFactory
from camayoc.ui.enums import SourceTypes

login_data = LoginFormDTOFactory()

credential_data = AddCredentialDTOFactory(credential_type=credential_type)

source_data = AddSourceDTOFactory(
    select_source_type__source_type=SourceTypes.SATELLITE,
    source_form__credentials=[credential_data.credential_form_dto.credential_name],
)
```

1.1.4 Design decisions

Here's the overview of framework goals, and brief explanation behind certain design decisions.

Leverage well-known Page Object Model design pattern Let's not reinvent the wheel. Anyone with experience in UI testing is familiar with Page Object Model. There are many articles, videos and tutorials explaining it. Make it easy for the next person.

The same drive towards familiarity is behind form design and choice of factory_boy and Faker.

Leverage static analysis to catch mistakes early UI tests have bad reputation because they tend to be slow, and that makes their feedback loop long. Simple mistake, like typo in input dictionary key, can easily cost 10 minutes, if it takes few minutes to run the test.

By heavy usage of type hints and strongly-typing method arguments, we hope that most of simple mistakes will be caught by editor, way before test is executed.

Make tests succinct Page objects "service methods" and DTO factories allow for writing relatively short tests.

Make common things easy... Most of tests are "happy-path tests", that only concern themselves with verifying that something may work in optimistic scenario. It should be easy and fast to write tests like these.

That's why page object methods focus on actual working actions and we don't provide methods for actions that result in invalid state of the system.

...and exceptional things possible At the same time, framework should not prevent test author from checking exceptions and reactions to invalid data.

That's why client exposes driver object. At any point of test, author can go to lower level and interact with page directly.

Allow for High Volume Automated Testing Most of tests verifies single, direct path through the system. Randomness, if any, is only in input data.

High Volume Automated Testing is the idea of writing tests that randomly journey through the system. Instead of having specific goal, they run for specified amount of time. They perform actions that should succeed, and inform the team when they encounter exceptions.

HiVAT are intended to find issues that become apparent only when using system for extended period of time - memory leaks, gradually worsening performance, unoptimized data access, assumptions about stored data size etc.

Framework is designed to make HiVAT possible. That's why we use a special form of Page Object Model, called Fluent Page Object Model. In Fluent POM, each method should return a page object. This makes it possible to chain method calls to model user journey through the system.

Some methods may require arguments. That's why framework expects all methods arguments to be type-hinted - so HiVAT could inspect method signature and use factories to generate required data. That's also why factories should generate data that is valid.

Proof of concept of HiVAT is implemented in `camayoc/tests/qpc/ui/test_long_running.py`.

1.2 OpenShift Tests

Caution: Work in Progress! This documentation is a place for describing how to test Quipucords inspection of OpenShift clusters (OCP) with Camayoc.

For running the Ansible Playbook to deploy a simple project to OpenShift, please make sure to have the OpenShift command-line interface (CLI) installed. Refer to the oc command for more information.

Verify if you have the CLI installed:

```
oc version
Client Version: 4.11.0-0.nightly-2022-05-18-171831
Kustomize Version: v4.5.4
Server Version: 4.10.43
Kubernetes Version: v1.23.12+8a6bfe4
```

CHAPTER 2

API Documentation

This is the Camayoc API documentation. It is mostly auto generated from the source code. This section of the documentation should be treated as a handy reference for developers, not a gospel.

2.1 camayoc package

A Python library that facilitates functional testing of quipucords.

2.1.1 Subpackages

camayoc.tests package

Tests for quipucords projects.

This package and its sub-packages contain functional tests for QPC. These tests should be run against respective QPC installations. These tests run remotely (trhough a SSH connection) or locally. These tests are entirely different from the unit tests in `tests`.

Subpackages

camayoc.tests.qpc package

Tests for Quipucords server.

This package and its sub-packages contain functional tests for quipucords server. These tests should be run against QPC installations.

See sub-modules for more information about requirements to run tests.

Subpackages

camayoc.tests.qpc.api package

Tests for Quipucords server.

This package and its sub-packages contain functional tests for quipucords server. These tests should be run against QPC installations.

QPC is interacted with over its API, and the base url should be specified in your camayoc config file. You can find a sample config file in the root of the repository named `example_config.yaml`.

Be sure to specify the port if you are running on a non-standard one.

Subpackages

camayoc.tests.qpc.api.v1 package

Tests for version 1 of the api.

Subpackages

camayoc.tests.qpc.api.v1.authentication package

Tests for Quipucords server authentication.

Submodules

camayoc.tests.qpc.api.v1.authentication.test_login module

camayoc.tests.qpc.api.v1.credentials package

Tests for Quipucords credentials.

Submodules

camayoc.tests.qpc.api.v1.credentials.test_creds_common module

camayoc.tests.qpc.api.v1.credentials.test_manager_creds module

camayoc.tests.qpc.api.v1.credentials.test_network_creds module

camayoc.tests.qpc.api.v1.reports package

Tests for Quipucords reports.

Submodules

camayoc.tests.qpc.api.v1.reports.test_reports module

camayoc.tests.qpc.api.v1.scanjobs package

Tests for Quipucords scanjobs.

Submodules

camayoc.tests.qpc.api.v1.scanjobs.test_pause_cancel_restart module

camayoc.tests.qpc.api.v1.scanjobs.test_run_scanjobs module

camayoc.tests.qpc.api.v1.scans package

Tests for Quipucords scans.

Submodules

camayoc.tests.qpc.api.v1.scans.test_scans_common module

camayoc.tests.qpc.api.v1.sources package

Tests for Quipucords sources.

Submodules

camayoc.tests.qpc.api.v1.sources.test_manager_sources module

camayoc.tests.qpc.api.v1.sources.test_network_sources module

camayoc.tests.qpc.api.v1.sources.test_sources_common module

Submodules

camayoc.tests.qpc.api.v1.conftest module

camayoc.tests.qpc.api.v1.utils module

camayoc.tests.qpc.cli package

Tests for quipucords' `qpc` command line interface.

Submodules

camayoc.tests.qpc.cli.conftest module

camayoc.tests.qpc.cli.csv_report_parsing module

Helper functions for parsing csv reports.

camayoc.tests.qpc.cli.csv_report_parsing.**extract_key_value_lines**(*input_lines*, *line_pairs*, *delim=', '*)

Extract multiple line pairs into list of dictionaries.

Parameters

- **input_lines** – a list of csv line strings.
- **line_pairs** – A list of tuples containg the index pairs for the lines to be used as the key/value pairs of each dictionary (ex: [(0,1), (3,4)]).
- **delim** – the delimiter to split the lines with (defaults to ‘,’).

camayoc.tests.qpc.cli.csv_report_parsing.**normalize_csv_report**(*f*, *header_range*, *header_lines*, *report_type='deployments'*)

Extract and normalize csv report to match the returned JSON report.

Parameters

- **f** – A file object for the csv
- **header_range** – An int specifing the range that the head extends into the csv file.
- **header_lines** – A list of tuples containg the index pairs for the lines to be used as the key/value pairs of each header info dictionary (ex: [(0,1), (3,4)]).
- **report_type** – A string that defines what type of report object to return, ‘deployments’ or ‘detail’. Defaults to ‘deployments’.

camayoc.tests.qpc.cli.csv_report_parsing.**normalize_deployments_report**(*header_info*, *reader*)

Normalize report info into a deployments report.

Takes information from report_info dict, and reader and returns a deployments report.

Parameters

- **header_info** – A list of dictonaries, which each dictonary containing the information of a header.
- **reader** – A DictReader containing the parsed content (remainder after header section) of a report.

camayoc.tests.qpc.cli.csv_report_parsing.**normalize_detail_report**(*header_info*, *reader*)

Normalize report info into a detail report.

Takes information from report_info dict, and reader and returns a detail format report

Parameters

- **header_info** – A list of dictonaries, which each dictonary containing the information of a header.

- **reader** – A DictReader containing the parsed content (remainder after header section) of a report.

camayoc.tests.qpc.cli.csv_report_parsing.**zip_line_pairs**(*input_lines*, *key_ind*, *value_ind*, *delim=', '*)

Take a list of csv strings, and combine 2 of them into a dictionary.

Parameters

- **input_lines** – a list of csv lines
- **key_ind** – the index value for which line in `input_lines` should be used for the keys of the dictionary.
- **value_ind** – the index value for which line in `input_lines` should be used for the values of the dictionary.
- **delim** – the delimiter to split the lines with (defaults to ',').

camayoc.tests.qpc.cli.test_credentials module

camayoc.tests.qpc.cli.test_openshift module

camayoc.tests.qpc.cli.test_reports module

camayoc.tests.qpc.cli.test_scanjobs module

camayoc.tests.qpc.cli.test_scans module

camayoc.tests.qpc.cli.test_sources module

camayoc.tests.qpc.cli.utils module

camayoc.tests.qpc.ui package

Tests for quipucords' UI.

Submodules

camayoc.tests.qpc.ui.conftest module

camayoc.tests.qpc.ui.test_credentials module

camayoc.tests.qpc.ui.test_login module

camayoc.tests.qpc.ui.test_long_running module

camayoc.tests.qpc.ui.test_sources module

camayoc.tests.qpc.ui.utils module

Utility class for UI tests.

camayoc.tests.qpc.ui.utils.**check_auth_type**(*credential_name*, *auth_type*)
Verify the authentication type of a credential.

Example types include 'SSH Key' and 'Username and Password'. If the Locator cannot find a match, an exception is raised.

camayoc.tests.qpc.ui.utils.**checkbox_xpath**(*row_name*)
Build an xpath for selecting a checkbox in a row.

Works for credentials or sources.

camayoc.tests.qpc.ui.utils.**clear_toasts**(*view*, *count=20*)
Attempt to flush any confirmation dialogs that may have appeared.

Use this function to clear out dialogs (toasts) that may be preventing buttons from being clicked properly. Sometimes it might need to be used in succession. By default, this tries to flush a maximum of 20 toasts, but will quit early if it cannot find more.

camayoc.tests.qpc.ui.utils.**create_credential**(*view*, *options*)
Create a credential through the UI.

camayoc.tests.qpc.ui.utils.**create_source**(*view*, *credential_name*, *source_type*, *source_name*, *addresses*)
Create a source through the UI.

camayoc.tests.qpc.ui.utils.**delete_credential**(*view*, *names*)
Delete a credential through the UI.

camayoc.tests.qpc.ui.utils.**delete_source**(*view*, *source_name*)
Delete a source through the UI.

camayoc.tests.qpc.ui.utils.**delete_xpath**(*row_name*)
Return an xpath for selecting the delete button in a row.

Works for credentials or sources.

camayoc.tests.qpc.ui.utils.**edit_credential**(*view*, *original_name*, *options*)
Edit a credential through the UI and verify it was edited.

Parameters

- **view** – The view context (should be the browser view)
- **original_name** – The original name of the credential.
- **options** – The options to be edited within the credential.

camayoc.tests.qpc.ui.utils.**edit_xpath**(*row_name*)
Return an xpath for selecting the edit button in a row.

Works for credentials or sources.

camayoc.tests.qpc.ui.utils.**field_xpath**(*label*, *textarea=False*)
Build an xpath for selecting a form field based on its label.

camayoc.tests.qpc.ui.utils.**fill**(*view*, *xpath_locator*, *text*)
Fill in a textbox using a xpath locator.

camayoc.tests.qpc.ui.utils.**fill_credential_info**(*view*, *options*)
Fill out the credential modal based on available options.

camayoc.tests.qpc.ui.utils.**get_field_value**(*view*, *label*, *textarea=False*)
Get the current value of a form field.

camayoc.tests.qpc.ui.utils.**radio_xpath**(*label*)
Build an xpath for selecting a radio button based on its label.

camayoc.tests.qpc.ui.utils.**row_xpath**(*row_name*)
Build an xpath for selecting a certain row.

Works for credentials or sources.

camayoc.tests.qpc.ui.utils.**set_checkbox**(*view*, *name*, *fill*)
Fill or clear a checkbox next to a credential.

camayoc.tests.qpc.ui.utils.**wait_for_animation**(*wait_time=0.5*)
Wait for animations to complete.

camayoc.tests.qpc.ui.views module

Quipucords views.

class camayoc.tests.qpc.ui.views.**CredentialModalView**(*parent*, *logger=None*, ***kwargs*)
Bases: *camayoc.tests.qpc.ui.views.ModalView*

Base class for credential modals.

save_button
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

class camayoc.tests.qpc.ui.views.**DashboardView**(*parent*, *logger=None*, ***kwargs*)
Bases: widgetastic.widget.base.View

Dashboard view.

logout
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

nav
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

user_dropdown
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

class camayoc.tests.qpc.ui.views.**DeleteModalView**(*parent*, *logger=None*, ***kwargs*)
Bases: *camayoc.tests.qpc.ui.views.ModalView*

Class for deletion dialogs.

delete_button
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

class camayoc.tests.qpc.ui.views.**LoginView**(*parent*, *logger=None*, ***kwargs*)
Bases: widgetastic.widget.base.View

Login view.

login
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

password
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

username
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

class camayoc.tests.qpc.ui.views.**ModalView**(*parent*, *logger=None*, ***kwargs*)
Bases: widgetastic.widget.base.View

Base class for modals.

cancel_button
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

class camayoc.tests.qpc.ui.views.**SourceModalView**(*parent*, *logger=None*, ***kwargs*)
Bases: *camayoc.tests.qpc.ui.views.ModalView*

Base class for source modals.

back_button
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

close_button
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

next_button
This class handles instantiating and caching of the widgets on view.

It stores the class and the parameters it should be instantiated with. Once it is accessed from the instance of the class where it was defined on, it passes the instance to the widget class followed by args and then kwargs.

It also acts as a counter, so you can then order the widgets by their "creation" stamp.

Submodules

camayoc.tests.qpc.conftest module

camayoc.tests.qpc.utils module

Submodules

camayoc.tests.conftest module

camayoc.tests.utils module

Utility functions.

camayoc.tests.utils.**get_vcenter_vms**(*vcenter_client*)
Return a list of available vCenter vms.

Parameters **vcenter_client** – A connected vCenter client which will be used to retrieve the list of all available vms.

camayoc.tests.utils.**is_live**(*cmd*, *server*, *num_pings=10*)
Test if server responds to ping.

Returns true if server is reachable, false otherwise.

camayoc.tests.utils.**is_vm_powered_on**(*vm*)
Check if a vCenter vm is powered on.

A vm is considered powered on when the VMWare Tools are installed, its power state is powered on and its IP address can be fetched.

camayoc.tests.utils.**power_off_vms**(*vms*, *timeout=300*)
Gracefully shutdown the vCenter vms.

camayoc.tests.utils.**power_on_vms**(*vms*, *timeout=300*)
Power on the vCenter vms.

For each virtual machine: ensure that the VMWare Tools are installed, they are powered on and its IP address can be fetched.

camayoc.tests.utils.**vcenter_vms**(*vms*)
Ensure vCenter machines are on and will be properly off.

Given a list of vCenter VM managed objects ensure all of them are on then yeild the vms list. Ensure they will be turned off before closing the context.

camayoc.tests.utils.**wait_until_live**(*servers*, *timeout=360*)
Wait for servers to be live.

For each server in the "servers" list, verify if it is reachable. Keep trying until a connection is made for all servers or the timeout limit is reached.

If the timeout limit is reached, we exit even if there are unreached hosts. This means tests could fail with "No auths valid for this profile" if every host in the profile is unreachable. Otherwise, if there is at least one valid host, the scan will go on and only facts about reached hosts will be tested.

camayoc.types package

Submodules

camayoc.types.settings module

camayoc.ui package

Subpackages

camayoc.ui.models package

Subpackages

camayoc.ui.models.pages package

Submodules

camayoc.ui.models.pages.abstract_page module

camayoc.ui.models.pages.credentials module

camayoc.ui.models.pages.login module

camayoc.ui.models.pages.scans module

camayoc.ui.models.pages.sources module

Submodules

camayoc.ui.models.fields module

camayoc.ui.models.mixins module

Submodules

camayoc.ui.client module

camayoc.ui.decorators module

camayoc.ui.enums module

class camayoc.ui.enums.**CredentialTypes**
Bases: *camayoc.ui.enums.StrEnum*

An enumeration.

NETWORK = 'Network'

SATELLITE = 'Satellite'

VCENTER = 'Vcenter'

class camayoc.ui.enums.**LowercasedStrEnum**
Bases: enum.Enum

An enumeration.

class camayoc.ui.enums.**MainMenuPages**
Bases: *camayoc.ui.enums.StrEnum*

An enumeration.

CREDENTIALS = 'Credentials'

SCANS = 'Scans'

SOURCES = 'Sources'

class camayoc.ui.enums.**NetworkCredentialAuthenticationTypes**
Bases: *camayoc.ui.enums.StrEnum*

An enumeration.

SSH_KEY = 'SSH Key'

USERNAME_AND_PASSWORD = 'Username and Password'

class camayoc.ui.enums.**NetworkCredentialBecomeMethods**
Bases: *camayoc.ui.enums.LowercasedStrEnum*

An enumeration.

DOAS = 'doas'

DZDO = 'dzdo'

KSU = 'ksu'

PBRUN = 'pbrun'

PFEXEC = 'pfexec'

RUNAS = 'runas'

SU = 'su'

SUDO = 'sudo'

class camayoc.ui.enums.**Pages**
Bases: *camayoc.ui.enums.StrEnum*

An enumeration.

CREDENTIALS = 'credentials.CredentialsMainPage'

LOGIN = 'login.Login'

SCANS = 'scans.ScansMainPage'

SOURCES = 'sources.SourcesMainPage'

SOURCES_RESULTS_PAGE = 'sources.ResultForm'

class camayoc.ui.enums.**SourceConnectionTypes**
Bases: *camayoc.ui.enums.StrEnum*

An enumeration.

DISABLE = 'Disable SSL'

SSL23 = 'SSLv23'

TLS1 = 'TLSv1'

TLS11 = 'TLSv1.1'

TLS12 = 'TLSv1.2'

class camayoc.ui.enums.**SourceTypes**
Bases: *camayoc.ui.enums.StrEnum*

An enumeration.

NETWORK_RANGE = 'network'

SATELLITE = 'satellite'

VCENTER_SERVER = 'vcenter'

class camayoc.ui.enums.**StrEnum**
Bases: enum.Enum

An enumeration.

camayoc.ui.session module

camayoc.ui.types module

2.1.2 Submodules

camayoc.api module

camayoc.command module

Execute local or remote commands.

class camayoc.command.**Command**(*system*, *response_handler=None*)
Bases: object

A convenience class for working with local or remote commands.

This class provides the ability to execute shell commands on either the local system or a remote system. Here is a pedagogic usage example:

```
>>> from camayoc import command
>>> system = command.System(hostname='localhost', transport='local')
>>> cmd = command.Command(system)
>>> response = cmd.run(('echo', '-n', 'foo'))
>>> response.returncode == 0
True
>>> response.stdout == 'foo'
True
>>> response.stderr == ''
True
```

The above example shows how various classes fit together. It's also verbose: smartly chosen defaults mean that most real code is much more concise.

You can customize how Command objects execute commands and handle responses by fiddling with the two public instance attributes:

machine A Plumbum machine. *run()* delegates all command execution responsibilities to this object.

response_handler A callback function. Each time machine executes a command, the result is handed to this callback, and the callback's return value is handed to the user.

If system.transport is local or ssh, machine will be set so that commands run locally or over SSH, respectively. If system.transport is None, the constructor will guess how to set machine by comparing the hostname embedded in system.hostname against the current system's hostname. If they match, machine is set to execute commands locally; and vice versa.

Parameters

- **system** (camayoc.command.System) – Information about the system on which commands will be executed.
- **response_handler** – A callback function. Defaults to *camayoc.command.code_handler()*.

run(*args*, ***kwargs*)
Run a command and return self.response_handler(result).

This method is a thin wrapper around Plumbum's BaseCommand.run method, which is itself a thin wrapper around the standard library's subprocess.Popen class. See their documentation for detailed usage instructions. See *camayoc.command.Command* for a usage example.

class camayoc.command.**CompletedProcess**(*args*, *returncode*, *stdout*, *stderr*)
Bases: object

A process that has finished running.

This class is similar to the `subprocess.CompletedProcess` class available in Python 3.5 and above. Significant differences include the following:

- All constructor arguments are required.
- *`check_returncode()`* returns a custom exception, not `subprocess.CalledProcessError`.

All constructor arguments are stored as instance attributes.

Parameters

- **`args`** – A string or a sequence. The arguments passed to *`camayoc.command.Command.run()`*.
- **`returncode`** – The integer exit code of the executed process. Negative for signals.
- **`stdout`** – The standard output of the executed process.
- **`stderr`** – The standard error of the executed process.

`check_returncode``()`

Raise an exception if `returncode` is non-zero.

Raise *`camayoc.exceptions.CalledProcessError`* if `returncode` is non-zero.

Why not raise `subprocess.CalledProcessError`? Because stdout and stderr are not included when str() is called on a CalledProcessError object. A typical message is:

```
"Command '('ls', 'foo')' returned non-zero exit status 2"
```

This information is valuable. One could still make `subprocess.CalledProcessError` work by overloading `args`:

```
>>> if isinstance(args, (str, bytes)):
...     custom_args = (args, stdout, stderr)
... else:
...     custom_args = tuple(args) + (stdout, stderr)
>>> subprocess.CalledProcessError(args, returncode)
```

But this seems like a hack.

In addition, it's generally good for an application to raise expected exceptions from its own namespace, so as to better abstract away dependencies.

class `camayoc.command.`**`System`**(*hostname*, *transport*)

Bases: `tuple`

A system representation to run commands on.

`hostname`

Alias for field number 0

`transport`

Alias for field number 1

`camayoc.command.`**`code_handler`**(*completed_proc*)

Check the process for a non-zero return code. Return the process.

Check the return code by calling `completed_proc.check_returncode()`. See: *`camayoc.command.CompletedProcess.check_returncode()`*.

`camayoc.command.`**`echo_handler`**(*completed_proc*)

Immediately return `completed_proc`.

camayoc.config module

camayoc.constants module

camayoc.data_provider module

camayoc.exceptions module

Custom exceptions defined by Camayoc.

exception camayoc.exceptions.**APIResultsEmpty**
Bases: *camayoc.exceptions.DataProviderException*

API request response results is empty.

This DataProvider is unable to yield any values.

exception camayoc.exceptions.**CalledProcessError**
Bases: Exception

Indicates a command process has a non-zero return code.

See *camayoc.command.CompletedProcess()* for more information.

exception camayoc.exceptions.**ConfigFileNotFoundError**
Bases: UserWarning

We cannot find the requested Camayoc configuration file.

See camayoc.config for more information on how configuration files are handled.

exception camayoc.exceptions.**DataProviderException**
Bases: Exception

Generic problem raised by DataProvider class

exception camayoc.exceptions.**FailedMergeReportException**
Bases: Exception

A test has raised this exception because a merge report job failed.

While waiting for the scan to acheive some other state, the merge report job failed. The test expected the merge report job to succeed, so this exception has been raised.

exception camayoc.exceptions.**FailedScanException**
Bases: Exception

A test has raised this exception because a scan failed.

While waiting for the scan to acheive some other state, the scan failed. The test expected the scan to succeed, so this exception has been raised.

exception camayoc.exceptions.**FilteredAPIResultsEmpty**
Bases: *camayoc.exceptions.DataProviderException*

There are no objects in API request response results matching provided criteria.

This configuration of DataProvider is unable to yield any values.

exception camayoc.exceptions.**IncorrectDecoratorUsageWarning**
Bases: UserWarning

Decorator was used incorrectly, but error is salvageable.

exception camayoc.exceptions.**MisconfiguredWidgetException**
Bases: Exception

Raised by UI Widget when expected property is not there.

exception camayoc.exceptions.**NoMatchingDataDefinitionException**
Bases: *camayoc.exceptions.DataProviderException*

Requested match_criteria do not match anything in DataProvider instance configuration.

exception camayoc.exceptions.**PageFactoryException**
Bases: Exception

Page factory received invalid value and can't instantiate new class.

exception camayoc.exceptions.**QPCBaseUrlNotFound**
Bases: Exception

Was not able to build a base URL with the config file information.

Check the expected configuration file format on the API Client documentation.

exception camayoc.exceptions.**StoppedScanException**
Bases: Exception

A test has raised this exception because a scan unexpectly stopped.

While waiting for the scan to achieve some other state, the scan reached a terminal state from which it could not progress, so this exception has been raised instead of continuing to wait.

exception camayoc.exceptions.**WaitTimeError**
Bases: Exception

A task has raised this error because it has been waiting too long.

A task was waiting for a long running task, but it has exceeded the time allowed. Instead of allowing the task to hang, it has aborted and raised this error.

camayoc.qpc_models module

camayoc.utils module

2.2 tests package

Unit tests for Camayoc.

This package and its sub-packages contain tests for Camayoc. These tests verify that Camayoc's internal functions and libraries function correctly.

2.2.1 Submodules

tests.test_api module

tests.test_command module

tests.test_data_provider module

tests.test_settings module

tests.test_utils module

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

c

t

Index

S

T

U

V

W

Z