pytest-dash Documentation

Release 2.1

Philippe Duval

Feb 22, 2019

Contents

CHAPTER 1

Usage

1.1 Install

Install with pip:

```
pip install -U pytest-dash
```

1.2 Write integration tests

`pytest-dash` provides fixtures and helper functions to write Dash integration tests.

To start a Dash instance, you can use a `dash_threaded` or `dash_subprocess` fixture.

The fixture will start the server when called and wait until the application has been loaded by the browser. The server will be automatically closed in the test teardown.

1.2.1 *dash_threaded* example

Start a dash application in a thread, close the server in teardown.

In this example we count the number of times a callback method is called each time a clicked is called and assert the text output of a callback by using *`wait_for_text_to_equal()`*.

```
import dash
from dash.dependencies import Output, Input
from dash.exceptions import PreventUpdate

import dash_html_components as html

from pytest_dash import wait_for

def test_application(dash_threaded):
```

(continues on next page)

(continued from previous page)

```
    # The selenium driver is available on the fixture.
    driver = dash_threaded.driver
    app = dash.Dash(__name__)
    counts = {'clicks': 0}

    app.layout = html.Div([
        html.Div('My test layout', id='out'),
        html.Button('click me', id='click-me')
    ])

    @app.callback(
        Output('out', 'children'),
        [Input('click-me', 'n_clicks')]
    )
    def on_click(n_clicks):
        if n_clicks is None:
            raise PreventUpdate

        counts['clicks'] += 1
        return 'Clicked: {}'.format(n_clicks)

    dash_threaded(app)

    btn = wait_for.wait_for_element_by_css_selector(driver, '#click-me')
    btn.click()

    wait_for.wait_for_text_to_equal(driver, '#out', 'Clicked: 1')
    assert counts['clicks'] == 1
```

1.2.2 *dash_subprocess* example

Start the server in subprocess with `waitress-serve`. Kill the process in teardown.

```
from pytest_dash.wait_for import wait_for_text_to_equal

def test_subprocess(dash_subprocess):
    driver = dash_subprocess.driver
    dash_subprocess('test_apps.simple_app')

    value_input = driver.find_element_by_id('value')
    value_input.clear()
    value_input.send_keys('Hello dash subprocess')

    wait_for_text_to_equal(driver, '#out', 'Hello dash subprocess')
```

Note: This fixture is slower than threaded due to the process spawning.

See also:

Fixtures dash_threaded *dash_threaded()*

dash_subprocess *dash_subprocess()*

1.2.3 Helpers

1.2.3.1 Importing applications

Import existing Dash applications from a file with *import_app()*. The application must be named app.

Example

```
from pytest_dash.application_runners import import_app

def test_application(dash_threaded):
    app = import_app('my_app')
    ...
```

1.2.3.2 Selenium wait for wrappers

The *wait_for* module is especially useful if you need to interact with elements or props that are only loaded after a callback as there might be a delay between when the callback is handled and returned to the frontend.

Available wrappers:

- *wait_for_element_by_css_selector()*
- *wait_for_elements_by_css_selector()*
- *wait_for_element_by_xpath()*
- *wait_for_elements_by_xpath()*
- *wait_for_element_by_id()*
- *wait_for_text_to_equal()*
- *wait_for_style_to_equal()*
- *wait_for_property_to_equal()*

1.3 Write declarative scenario tests

Pytest-dash include a declarative way to generate tests in a yaml format. When pytest finds yaml files prefixed with `test_` in a directory, it will generate tests from a `Tests` object.

1.3.1 Schema

A yaml test file contains scenario definitions and a list of parametrized of scenarios to execute.

1.3.1.1 Globals

application Global default application to use in the tests if no option supplied.

Tests List of scenario to generate tests for. Test item props are used as parameter.

1.3.1.2 Scenario object

parameters Object where the keys will be used to create a variable dictionary to use in behavior commands. Use a parameter in commands by prefixing the key with `$`, (eg: `$value`).

application

path Dot notation path to the application file.

options

port The port used by the application.

event List of actions to execute.

outcome List of expected result of the scenario event.

Listing 1: Commented example

```
Scenario:           # Key of the test
    parameters:     # Optional values to use in test
        value:
            default: 4
    application:    # The application settings to use in the test
        path: test_apps.simple_app  # Dot notation path to the app file.
        options:    # Application options such as port
            port: 8051
    event:          # List of actions describing what happen.
        - "enter $value in #input"
    outcome:        # The expected result of the event.
        - "text in #output should be $value"

Tests:              # List of all the scenarios to execute.
    - Scenario      # Runs Scenario with the default parameter.
    - Scenario
        value: 8    # Override the default parameter.
```

1.3.2 Syntax

There is 3 kind of rule for the grammar:

- `value`, return a value.
- `command`, execute an action.
- `comparison`, compare two value.

Table 1: Scenario event/outcome syntax

Rule	Kind	Example	Description
element_id	value	`#my-element-id`	Find a single element by id
element_selector	value	`{#my-element-id > span}`	Find a single by selector
elements_selector	value	`*{#my-element-id > span}`	Find multiple elements by selector, actions will be executed on all elements (Currently click & length assertions)
element_xpath	value	`[//*[@id="btn-1"]]`	Find a single element by xpath
elements_xpath	value	`*[//div[@id="container"]/span]`	Find multiple elements by xpath.
element_prop	value	`#my-input.value`	A property of an element to use in comparisons.
eq	comparison	`#my-input.value should be 1`, `#my-input.value == 1`	Equality comparison
lt	comparison	`#my-input.value < 3`, `#my-input.value should be less than 3`	The value should be less than.
lte	comparison	`#my-input.value <= 3`,‘‘#my-input.value should be less or equal than 3‘‘	The value on the left should be less or equal to.
gt	comparison	`#my-input.value > 3`, `#my-input.value should be greater than 3`	Value should be greater.
gte	comparison	`#my-input.value >= 3`, `#my-input.value should be greater or equal than 3`	Greater or equal comparison.
text_equal	comparison	`text in #output should be "Foo bar"`	Special comparison for text attribute, it uses the `wait_for` api.
prop_comparison	comparison	`#output.value should be 3`	Property comparison uses the `wait_for` api
style_comparison	comparison	`style "padding" of #btn should be "3px"`	`wait_for` comparison for a style attribute of an element.
clear	command	`clear #my-input`	Clear the value of an element.
click	command	`click #my-btn`	Click on an element, the element must be visible to be clickable.
send_value	command	`enter "Foo bar" in #my-input`	Send keyboard input to an element.

Note: The syntax can be extended with *Hooks*.

1.3.3 Examples

Application

Listing 2: simple_app.py

```
import dash
from dash.dependencies import Output, Input
from dash.exceptions import PreventUpdate

import dash_html_components as html
import dash_core_components as dcc

app = dash.Dash(__name__)

app.layout = html.Div([
    dcc.Input(id='value', placeholder='my-value'),
    html.Div(['You entered: ', html.Span(id='out')])
])


@app.callback(Output('out', 'children'), [Input('value', 'value')])
def on_value(value):
    if value is None:
        raise PreventUpdate

    return value
```

Test

Listing 3: test_simple_callback.yml

```
SimpleCallback:
    description: Test a dcc.Input callback output to a html.Div when a html.Button is␣
↪clicked\
    parameters:
        var1:
            description: Value to send to the input
            type: str
            default: hello world
    application:
        path: test_apps.simple_app
        port: 8051
    event:
        - "clear #value"
        - "enter $var1 in #value"
    outcome:
        - "#value.value == $var1"
        - 'text in {#out} should be $var1'

Tests:
    - SimpleCallback
    - SimpleCallback:
        var1: foo bar
```

See also:

Component gallery behavior test

1.4 Run tests

Use `$ pytest tests --webdriver Chrome` to run all the test

The `--webdriver` option is used for choosing the selenium driver to use. Choose from:

- Chrome
- Firefox
- Safari
- Edge
- Opera
- PhantomJS
- Ie
- Remote

Note: The driver must be available on your environment *PATH*.

See also:

Please refer to https://selenium-python.readthedocs.io/installation.html for selenium installation.

1.4.1 Configuration

The default webdriver for a project can be specified in pytest.ini instead of having to enter it on the command line every time you run a test.

Example `./pytest.ini`

```
[pytest]
webdriver = Chrome
```

1.4.2 Hooks

1.5 pytest_add_behaviors

The scenario event/outcome syntax can be extended with the *`pytest_add_behaviors()`* hook.

`add_behavior` is a decorator with the following keywords arguments:

- `syntax` The syntax to match, it will be available under the name of the function in the parser.
- **`kind`**
  - `value` default, A value can be used in commands and comparisons.
  - `command`, Complete custom line parsing.
  - `comparison`, A comparison should `assert` something inside the function.
- `inline/tree/meta` Only one can to be set to true, default is inline, decorate the function with `lark.v_args(inline=inline, tree=tree, meta=meta)`, lark.v_args docs.

Example `tests/conftest.py`

```
def pytest_add_behaviors(add_behavior):
    @add_behavior('"eval("/.*/")"')
    def evaluate(command):
        return eval(command)
```

See also:

Lark grammar reference https://lark-parser.readthedocs.io/en/latest/grammar/

1.6 pytest_setup_selenium

If you need to configure the selenium driver used by the plugin, you can use the `pytest_setup_selenium` hook.

Example `tests/conftest.py`

Listing 4: Run chrome in headless mode.

```
from selenium.webdriver.chrome.options import Options


def pytest_setup_selenium(driver_name):
    options = Options()
    options.headless = True
    return {
        'chrome_options': options,
    }
```

CHAPTER 2

pytest_dash

2.1 pytest_dash package

2.1.1 Submodules

2.1.2 pytest_dash.application_runners module

Run dash applications with a context manager. When exiting the context, the server will close.

class pytest_dash.application_runners.**BaseDashRunner**(*driver*, *keep_open=False*)
Bases: object

Base context manager class for running applications.

__init__(*driver*, *keep_open=False*)

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **keep_open** (*bool*) – Keep the server open

start(**args*, ***kwargs*)
Start the application.

Parameters

- **args** –
- **kwargs** –

Returns

stop()
Stop the dash application.

Returns

url
The url with the port auto-formatted.

Returns Formatted url with the port.

class pytest_dash.application_runners.**DashSubprocess**(*driver*, *keep_open=False*)
Bases: *pytest_dash.application_runners.BaseDashRunner*

Runs a dash application in a waitress-serve subprocess.

__init__(*driver*, *keep_open=False*)

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **keep_open** (*bool*) – Keep the server open

start(*app_module*, *application_name='app'*, *port=8050*)
Start the waitress-serve process.

See also:

dash_subprocess()

Parameters

- **app_module** (*str*) – Dot notation path to the app file.
- **application_name** (*str*) – Variable name of the dash instance.
- **port** (*int*) – Port to serve the application.

Returns

stop()
Stop the dash application.

Returns

class pytest_dash.application_runners.**DashThreaded**(*driver*, *keep_open=False*)
Bases: *pytest_dash.application_runners.BaseDashRunner*

Runs a dash application in a thread.

__init__(*driver*, *keep_open=False*)

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **keep_open** (*bool*) – Keep the server open

start(*app*, *port=8050*, *start_wait_time=0.5*, *start_timeout=10*, ***kwargs*)
Start the threaded dash app server.

See also:

dash_threaded()

Parameters

- **app** (*dash.Dash*) – The dash application instance.
- **port** (*int*) – Port of the dash application.

- **start_wait_time** (*float*) – Poll rate for the server started wait
- **start_timeout** (*float*) – Max time to start the server.
- **kwargs** –

Returns

stop()
Stop the dash application.

Returns

pytest_dash.application_runners.**import_app**(*app_file*, *application_name='app'*)
Import a dash application from a module. The import path is in dot notation to the module. The variable named app will be returned.

Example

```
>>> app = import_app('my_app.app')
```

Will import the application in module *app* of the package *my_app*.

Parameters

- **app_file** (*str*) – Path to the app (dot-separated).
- **application_name** – The name of the dash application instance.

Raise pytest_dash.errors.NoAppFoundError

Returns App from module.

Return type dash.Dash

2.1.3 pytest_dash.behavior_parser module

Custom lark parser and transformer for dash behavior tests.

class pytest_dash.behavior_parser.**BehaviorTransformer**(*driver*, *variables=None*)
Bases: lark.visitors.Transformer, object

Transform and execute behavior commands.

__init__(*driver*, *variables=None*)

Parameters **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver to find elements in the tree

clear(**args*, ***kwargs*)
Clear an element.

Example `clear #element`

Kind command

click(**args*, ***kwargs*)
Click an element.

Example `click #element`

Kind command

compare(**args*, ***kwargs*)

element(**args*, ***kwargs*)

element_id(**args*, ***kwargs*)
Find an element by id when found in the tree.

Example `#dropdown`

Kind value

Parameters **element_id** – Text after *#*

element_prop(**args*, ***kwargs*)
Property value of an element

Example `#element.prop`

Kind value

element_selector(**args*, ***kwargs*)
Find an element by selector when found in the tree.

Example `{#radio-items > label:nth-child(9) > input[type="radio"]}`

Kind value

Parameters **selector** – Text contained between *{* & *}*

element_xpath(**args*, ***kwargs*)
Find an element by xpath

Example `[//div/span]`

Kind value

elements(**args*, ***kwargs*)

elements_length(**args*, ***kwargs*)

elements_selector(**args*, ***kwargs*)

elements_xpath(**args*, ***kwargs*)
Find all elements by xpath

Example `*[//div/span]`

Kind value

escape_string(**args*, ***kwargs*)
Escaped string handler, remove the `"` from the token.

Kind value

false_value(**args*, ***kwargs*)

number(**args*, ***kwargs*)

prop_compare(**args*, ***kwargs*)
Wait for a property to equal a value.

Example `#output.id should be "my-element"`

Kind comparison

select_by_index(**args*, ***kwargs*)

select_by_text(**args*, ***kwargs*)

select_by_value(**args*, ***kwargs*)

send_value (**args*, ***kwargs*)
Send key inputs to the element

Example `enter "Hello" in #input`

Kind command

style_compare (**args*, ***kwargs*)
Compare a style value of an of element.

Example style "color" of #style should be "rgba(0, 0, 255, 1)"

Kind comparison

Parameters

- **style** – Name of the style property
- **element** – Element to find
- _ – eq
- **value** – Value to compare to the element style attribute.

Returns

text_equal (**args*, ***kwargs*)
Assert the text attribute of an element is equal with a wait timer.

Example `text #output should be "Foo bar"`

Kind comparison

true_value (**args*, ***kwargs*)

variable (**args*, ***kwargs*)
A variable specified in the parameters attribute of behavior.

Example

```
ValueBehavior:
  parameters:
    value:
        default: Foo
  event:
    - "enter $value in #input"
  outcome:
    - "text in #input-output should be $value"

Tests:
    ValueBehavior
    ValueBehavior:
        - value: Bar
```

Kind value

class pytest_dash.behavior_parser.**BehaviorTransformerMeta**
Bases: `type`

Dynamically create a parser transformer with user defined behaviors

pytest_dash.behavior_parser.**parser_factory** (*driver*, *variables=None*, *behaviors=None*)
Create a Lark parser with a BehaviorTransformer with the provided selenium driver to find the elements.

A new behavior transformer class is created and behaviors are assigned a transformer function from the supplied behaviors in the pytest_add_behaviors hook.

Parameters

- **driver** – Selenium driver to use when parsing elements.
- **variables** – Variables to use in the parser transformer.
- **behaviors** – Custom behaviors, come from plugin.behaviors.

Returns

2.1.4 pytest_dash.behaviors module

Experimental behavioral test api for dash apps.

class pytest_dash.behaviors.**DashBehaviorTestFile**(*fspath*, *parent*, *plugin*)
Bases: _pytest.nodes.File

A yaml test file definition

__init__(*fspath*, *parent*, *plugin*)
x.__init__(...) initializes x; see help(type(x)) for signature

collect()
returns a list of children (items and collectors) for this collection node.

class pytest_dash.behaviors.**DashBehaviorTestItem**(*plugin*, *name*, *parent*, *spec*, *application=None*, ***kwargs*)
Bases: _pytest.nodes.Item

A single test of a test file.

__init__(*plugin*, *name*, *parent*, *spec*, *application=None*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

reportinfo()

runtest()

2.1.5 pytest_dash.errors module

Pytest-dash errors.

exception pytest_dash.errors.**DashAppLoadingError**
Bases: *pytest_dash.errors.PytestDashError*

The dash app failed to load

exception pytest_dash.errors.**InvalidDriverError**
Bases: *pytest_dash.errors.PytestDashError*

An invalid selenium driver was specified.

exception pytest_dash.errors.**MissingBehaviorError**
Bases: *pytest_dash.errors.PytestDashError*

A behavior was missing from the

exception pytest_dash.errors.**NoAppFoundError**
Bases: *pytest_dash.errors.PytestDashError*

No *app* was found in the file.

exception pytest_dash.errors.**PytestDashError**
Bases: exceptions.Exception

Base error for pytest-dash.

exception pytest_dash.errors.**ServerCloseError**
Bases: *pytest_dash.errors.PytestDashError*

Pytest-dash had trouble closing a server.

2.1.6 pytest_dash.new_hooks module

Custom hooks for pytest dash

pytest_dash.new_hooks.**pytest_add_behaviors**(*add_behavior*)
Use this hook to add custom behavior parsing.

Example *conftest.py*

```
def pytest_add_behavior(add_behavior):
    @add_behavior('Text to parse')
    def custom_parse_action(params):
        pass
```

Parameters **add_behavior** – Decorator for a behavior handler function.

Returns

pytest_dash.new_hooks.**pytest_setup_selenium**(*driver_name*)
Called before the driver is created, return a dictionary to use as kwargs for the driver init.

Parameters **driver_name** (*str*) – The name of the driver specified by either cli argument or in pytest.ini.

Returns The dictionary of kwargs to give to the driver constructor.

2.1.7 pytest_dash.plugin module

2.1.7.1 Pytest-dash plugin

Main entry point for pytest

- Hooks definitions
- Plugin config container
- Plugin selenium driver
- Fixtures

class pytest_dash.plugin.**DashPlugin**
Bases: object

Plugin configuration and selenium driver container

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

driver

pytest_collect_file(*parent*, *path*)

pytest_configure(*config*)

pytest_unconfigure(*config*)

pytest_dash.plugin.**dash_subprocess**(**args*, ***kwargs*)
Start a Dash server with subprocess.Popen and waitress-serve.

Example

```
def test_application(dash_subprocess):
    # consider the application file is named `app.py`
    dash_subprocess('app')
```

See also:

pytest_dash.application_runners.DashSubprocess

pytest_dash.plugin.**dash_threaded**(**args*, ***kwargs*)
Start a local dash server in a new thread. Stop the server in teardown.

Example

```
import dash
import dash_html_components as html

def test_application(dash_threaded):
    app = dash.Dash(__name__)
    app.layout = html.Div('My app)
    dash_threaded(app)
```

See also:

pytest_dash.application_runners.DashThreaded

pytest_dash.plugin.**pytest_addhooks**(*pluginmanager*)

pytest_dash.plugin.**pytest_addoption**(*parser*)

pytest_dash.plugin.**pytest_configure**(*config*)

2.1.8 pytest_dash.wait_for module

Utils methods for pytest-dash such wait_for wrappers

pytest_dash.wait_for.**wait_for_element_by_css_selector**(*driver*, *selector*, *timeout=10.0*)
Wait until a single element is found and return it. This variant use the css selector api: https://www.w3schools.com/jsref/met_document_queryselector.asp

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **selector** (*str*) – CSS selector to find.
- **timeout** (*float*) – Maximum time to find the element.

Returns

pytest_dash.wait_for.**wait_for_element_by_id**(*driver*, *_id*, *timeout=10*)

Wait until a single element is found and return it. This variant find by id.

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **_id** – The id of the element to find.
- **timeout** (*float*) – Maximum time to find the element.

Returns

pytest_dash.wait_for.**wait_for_element_by_xpath**(*driver*, *xpath*, *timeout=10*)

Wait until a single element is found and return it. This variant use xpath to find the element. https://www.w3schools.com/xml/xml_xpath.asp

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **xpath** (*str*) – Xpath query string.
- **timeout** (*float*) – Maximum time to find the element.

Returns

pytest_dash.wait_for.**wait_for_elements_by_css_selector**(*driver*, *selector*, *timeout=10.0*)

Wait until all elements are found by the selector before the timeout.

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **selector** (*str*) – Search for elements
- **timeout** (*float*) – Maximum wait time

Returns Found elements

pytest_dash.wait_for.**wait_for_elements_by_xpath**(*driver*, *xpath*, *timeout=10*)

Wait until all are found before the timeout. This variant use xpath to find the elements. https://www.w3schools.com/xml/xml_xpath.asp

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **xpath** (*str*) – Xpath query string.
- **timeout** (*float*) – Maximum time to find the element.

Returns

pytest_dash.wait_for.**wait_for_property_to_equal**(*driver*, *selector*, *prop_name*, *prop_value*, *timeout=10*)

Wait for an element property to equal a value.

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **selector** (*str*) – Selector of the element to assert it's property.
- **prop_name** (*str*) – The name of property.
- **prop_value** – The value to assert.
- **timeout** (*float*) – Maximum time.

Returns

pytest_dash.wait_for.**wait_for_style_to_equal**(*driver*, *selector*, *style_attribute*, *style_assertion*, *timeout=10*)

Wait for an element style attribute to equal.

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **selector** (*str*) – Selector of the element to assert it's style property.
- **style_attribute** (*str*) – The name of the CSS attribute to assert.
- **style_assertion** (*str*) – The value to equal of CSS attribute.
- **timeout** (*float*) – Maximum time.

Returns

pytest_dash.wait_for.**wait_for_text_to_equal**(*driver*, *selector*, *text*, *timeout=10*)

Wait an element text found by css selector is equal to text.

Parameters

- **driver** (*selenium.webdriver.remote.webdriver.WebDriver*) – Selenium driver
- **selector** (*str*) – Selector of the element to assert it's text property.
- **text** (*str*) – Text to equal.
- **timeout** (*float*) – Maximum time for the text to equal.

Returns

2.1.9 Module contents

2.1.9.1 Pytest-dash

Pytest fixtures and helper methods for dash.

CHAPTER 3

Examples

3.1 *dash_threaded* example

Start a dash application in a thread, close the server in teardown.

In this example we count the number of times a callback method is called each time a clicked is called and assert the text output of a callback by using *wait_for_text_to_equal()*.

```
import dash
from dash.dependencies import Output, Input
from dash.exceptions import PreventUpdate

import dash_html_components as html

from pytest_dash import wait_for

def test_application(dash_threaded):
    # The selenium driver is available on the fixture.
    driver = dash_threaded.driver
    app = dash.Dash(__name__)
    counts = {'clicks': 0}

    app.layout = html.Div([
        html.Div('My test layout', id='out'),
        html.Button('click me', id='click-me')
    ])

    @app.callback(
        Output('out', 'children'),
        [Input('click-me', 'n_clicks')]
    )
    def on_click(n_clicks):
        if n_clicks is None:
            raise PreventUpdate
```

(continues on next page)

(continued from previous page)

```
        counts['clicks'] += 1
        return 'Clicked: {}'.format(n_clicks)

    dash_threaded(app)

    btn = wait_for.wait_for_element_by_css_selector(driver, '#click-me')
    btn.click()

    wait_for.wait_for_text_to_equal(driver, '#out', 'Clicked: 1')
    assert counts['clicks'] == 1
```

3.2 *dash_subprocess* example

Start the server in subprocess with `waitress-serve`. Kill the process in teardown.

```
from pytest_dash.wait_for import wait_for_text_to_equal

def test_subprocess(dash_subprocess):
    driver = dash_subprocess.driver
    dash_subprocess('test_apps.simple_app')

    value_input = driver.find_element_by_id('value')
    value_input.clear()
    value_input.send_keys('Hello dash subprocess')

    wait_for_text_to_equal(driver, '#out', 'Hello dash subprocess')
```

Note: This fixture is slower than threaded due to the process spawning.

3.3 Component gallery behavior test

Application

Listing 1: test_apps/component_gallery.py

```
import dash
from dash.dependencies import Output, Input
from dash.exceptions import PreventUpdate

import dash_html_components as html
import dash_core_components as dcc

app = dash.Dash(__name__)

app.layout = html.Div([
    dcc.Input(id='input'),
    dcc.Dropdown(
        id='dropdown',
        options=[{
```

(continues on next page)

(continued from previous page)

```
            'label': str(x),
            'value': str(x)
        } for x in range(1, 10)]
    ),
    dcc.RadioItems(
        id='radio-items',
        options=[{
            'label': str(x),
            'value': str(x)
        } for x in range(1, 10)]
    ),
    html.Div(id='input-output'),
    html.Div(id='dropdown-output'),
    html.Div(id='radio-items-output'),
    html.Button('change-style', id='change-style'),
    html.Div(
        'Style changer',
        id='style-output',
        style={'backgroundColor': 'rgba(255, 0, 0, 1)'}
    ),
    html.Div(
        id='multi-elements-click',
        children=[
            html.Button('btn-1', id='btn-1'),
            html.Button('btn-2', id='btn-2'),
            html.Button('btn-3', id='btn-3'),
        ]
    ),
    html.Div(id='multi-elements-outputs')
])

for i in (
        'input',
        'dropdown',
        'radio-items',
):

    @app.callback(
        Output('{}-output'.format(i), 'children'), [Input(i, 'value')]
    )
    def _wrap(value):
        if value is None:
            raise PreventUpdate

        return str(value)


@app.callback(
    Output('style-output', 'style'), [Input('change-style', 'n_clicks')]
)
def on_style_change(n_clicks):
    if n_clicks is None:
        raise PreventUpdate

    return {'backgroundColor': 'rgba(0, 0, 255, 1)'}
```

(continues on next page)

(continued from previous page)

```
@app.callback(
    Output('multi-elements-outputs', 'children'),
    [Input('btn-{}'.format(x), 'n_clicks') for x in range(1, 4)],
)
def on_multi_click(*args):
    return [html.Span(x) for x in args if x]
```

Test

Listing 2: tests/test_component_gallery.yml

```
application:
  path: test_apps.component_gallery

InputBehavior:
  description: Test text can entered/customized in a Input component in behavior␣
↪tests.
  parameters:
    value:
        default: Input value
  event:
    - "enter $value in #input"
  outcome:
    - "text in #input-output should be $value"

DropdownBehavior:
  description: Test that a dropdown value can be selected in behavior tests.
  parameters:
    value:
      default: '1'
  event:
    - 'enter $value in {#dropdown .Select-input input}'
    - 'click {#dropdown .Select .Select-menu-outer}'
  outcome:
    - 'text in #dropdown-output should be $value'

RadioItemsBehavior:
  description: Test radio items value can be changed in behavior tests.
  event:
    - 'click {#radio-items > label:nth-child(9) > input[type="radio"]}'
  outcome:
    - "text in #radio-items-output should be 9"
    - '{#radio-items > label:nth-child(9) > input[type="radio"]}.checked should be␣
↪true'

StyleChangeButtonBehavior:
  description: Test style assertions in behavior tests
  event:
    - 'style "background-color" of #style-output should be "rgba(255, 0, 0, 1)"'
    - 'click #change-style'
  outcome:
    - 'style "background-color" of #style-output should be "rgba(0, 0, 255, 1)"'

MultiElementsBehavior:
  description: Test multiple elements found.
  event:
    - 'click *{#multi-elements-click > button}'
```

(continues on next page)

(continued from previous page)

```
  outcome:
    - '*{#multi-elements-outputs > span}.length == 3'

XpathBehavior:
  description: Test xpath element find.
  event:
    - 'click [//*[@id="btn-1"]]'
  outcome:
    - '*[//div[@id="multi-elements-outputs"]/span].length should be 1'

Tests:
  - InputBehavior
  - InputBehavior:
      value: Non default
  - DropdownBehavior
  - RadioItemsBehavior
  - StyleChangeButtonBehavior
  - MultiElementsBehavior
  - XpathBehavior
```

Python Module Index

p

Index

Symbols

B

C

D

E

F

I