Behave Manners Documentation

Release 0.10.0

Panos Christeas

Oct 11, 2019

Contents

Topics:

CHAPTER 1

Manners Overview

behave_manners is designed to be a weapon against complexity of modern web sites (rather, applications). Offers utility methods and a powerful abstraction layer on top of *Web Elements*, the standard objects of WebDriver API.

'manners' offers tools in these areas:

1. setting up the 'site' and calling the client browser
2. navigating through the site, by URLs and matching 'known pages'
3. decoding the remote DOM into an abstract structure of *components*
4. consistent error handling, logs and screenshots as a separate layer

Also, some early work is done in enriching the *Gherkin* language with more structure, re-usable scenarios and resources.

CHAPTER 2

Introduction to Page Elements

Page elements (or pagelems) is a custom html-like language of defining patterns for **matching** remote DOM trees. This language is designed to resemble HTML as much as possible, but is NOT the kind of html that would render into the DOM of a page.

Rather, it is the inverse: it is a declarative language of *parsing* that DOM.

2.1 Rules/principles

Understanding the design of page elements comes after considering the following:

Principles

- Markup shall be as simple as possible
- Markup must resemble HTML, share concepts/features of HTML5
- developer should write enough markup to uniquely identify components
- Markup shall provide a way to *mark* components discovered, and their attributes
- Markup is by design nested, a lot
- Markup shall provide ways to share blocks and refer back and forth to others

Rules

1. plain markup should match the same markup on the remote eg. `<div class="foo"><span>Text</span></div>` should match a DIV containing a span with text=”Text” etc.
2. attributes and elements not mentioned DO match In the example above, all other attributes of the `<div>` or some other element within that should be allowed.
3. children elements match children in the DOM, *direct* descendants unless explicitly decorated (with `pe-deep`)
4. some special elements+attributes, all decorated with the **pe-** prefix may modify this straightforward HTML matching

5. the special `this` attribute defines a component. All other elements are just matched, harvested for attributes, but not exposed.
6. attributes defined multiple times may be OR-ed together

2.2 Special elements

Define constructs that modify the way DOM is structured.

`<pe-repeat>`

Causes its contents to be matched multiple times, repeated naturally.

Note: this is implied if `this` has a flexible definition, see below.

`<pe-choice>`

Attempts to match the first of the contained components. Any of them may match.

`<pe-any>`

Matches any html element. Otherwise its attributes and children match like any plain element.

`<pe-data>`

Assigns arbitrary data as an attribute to current component. Like a constant to some programming language. May serve as customization data for re-usable controllers on that component.

`<pe-regex>`

Matches element text by regular expression; then assigns attributes from named members of that expression

`<pe-deep>`

Matches (its children) at any level deep from the parent element

`<pe-root>`

Resets nesting to the DOM document root, matches children down from the root

`<pe-slotcontent>`

Points back to the original content of the *slot*. See *Templates and Slots*

`<pe-group>`

Matches all of the contained elements, in order. *pe-group* itself needs not match any element. Useful for `<pe-choice>` and repetitions

2.3 Special attributes

`pe-deep`

Matches this element any level down from the parent, not just direct ancestors

`pe-optional`

Makes this element (and all contents) optional. No error if cannot match

`pe-controller` `pe-ctrl`

Picks an alternate controller, defines a scope at that level. See *Scopes and Controllers*

`slot`

Defines slot name. See *Templates and Slots*

2.4 Special attribute: this

`this` deserves a section of its own.

It defines a component in the matched tree.

In its simplest form: `this="egg"` , it would define a component named "egg" In the attribute form: `this="[title]"` it will scan the contents, then resolve the *title* attribute, use the title's value as a name. In the parametric form: `this="col_%d"` it may produce more than one components using the integer count of matches.

Example:

```
<div class="chapter" this="first-chapter">
    <section this="[title]">
        <h3>[title]</h3>

        <p this="par_%d">
            [content]
        </p>
    </section>
</div>
```

The above would produce a Component tree like:

```
first-chapter/
    Some Title/
        par_0/
          content="..."
        par_1/
          content="..."
    Other Title/
        par_0/
          content="..."
```

Which exposes, say, that second content in python as:

```
root['first-chapter']['Some Title']['par_1'].content
```

2.5 Templates and Slots

Templates are described in HTML5, are a way for the browser to repeat rendering some block of HTML in multiple places of the DOM. Likewise, in pagelements, they allow blocks of pagelem markup to be re-used across the page or any components.

Example:

```
<template id="complete-text-field">
    <div class="field-container" this="[name]">
        <div>
```

(continues on next page)

(continued from previous page)

```
            <label>[field]</label>
            <input pe-deep type="text" name="[name]" this="input">
        </div>
    </div>
</template>
```

then:

```
<form this="the-form">
    <div class="form-container">
        <!-- matches all text fields, indexes by their `name` -->
        <use-template id="complete-text-field"/>

        <!-- matches that `name="size"` number field, explicitly -->
        <div class="field-container" this="the-size">
            <input pe-deep type="number" name="size" this="input"/>
        </div>
    </div>
</form>
```

Templates can be defined in the *<head>* of the pagelem html, in the *<body>* or in separate *gallery* files, from where they can be re-used.

Templates can have custom content, per call, that will be substitued in the pagelem markup:

```
<template id="custom-text-field">
    <div class="field-container" this="[name]">
        <div>
            <slot name="label">
                <label>[field]</label>
            </slot>
            <input pe-deep type="text" name="[name]" this="input">
        </div>
    </div>
</template>

<form>
    <use-template id="custom-text-field">
        <!-- matches that label only -->
        <label slot="label">Just this one</label>
    </use-template>

    <use-template id="custom-text-field">
        <span slot="label">Odd field</span> <!-- this uses different element -->
    </use-template>
</form>
```

CHAPTER 3

Scopes and Controllers

Scopes and Controllers are a way to inject extra behaviour onto components, from code written in Python. They are able to abstract non-trivial interactions as if they were instance methods to the component objects.

Scopes have their own tree (hierarchy) are deliberately *not* 1:1 with components. Reason is to keep code minimal, not to mandate extra code per each component. In practice, few of the components, only, will ever need their own controller.

Note: *scopes* and *controllers* are used as terms interchangeably, they are the same thing. A *controller* is the class that defines custom behaviour, a *scope* is the instantiated object of that class, that could also have state.

3.1 Service Metaclass

Controller classes are using the *Service Meta* class, derive from `DOMScope` . This means that they can be referenced by `_name`, and can be overriden by defining the same name in some custom python module.

3.2 Inheritance

Apart from python (or rather service-meta) inheritance, scopes also hold a reference of their parent in the Component Tree. By default, they *inherit* the attributes of that parent scope. This allows nested components to call parent scope's methods, or even to modify that parent's state.

In the example below, suppose that there is a minimal HTML document (into components) with one component calling a custom controller (the 'fld-ctrl' one).

In python, suppose the following code defines the controller classes:

```
class MyScope(DOMScope):
    _name = 'page'
```

(continues on next page)

(continued from previous page)

```
    def hello(self):
        return "Hello!"

class FldCtrl(DOMScope):
    _name = 'fld-ctrl'

    def check_field(self, field):
        return field.value == 'ham'
```

Scope inheritance would look like this diagram:

There is always a root scope (usually the `.root` class) [1] and a page one[2]. Also, all scopes inherit `DOMScope` which is denoted by `[*]`. But our `MyScope` class redefines the page, so it would have methods from both 'page' classes. So, anywhere in the document, components could say:

```
content._scope.hello()
```

and use that method. Including the 'field' component.

Then, the field component has its own controller [3], hence scope. This means that the field component, including any sub-components there, can say:

```
field_component._scope.check_field(sub_component)
```

3.3 Controllers of components

But calling `_scope` of a component, then passing the component as an argument is not trivial, is it? In fact, by design, this is meant only for some inter-component methods.

Component methods can be defined within a controller class through the **Component** special class:

```
class FldCtrl(DOMScope):
    _name = 'fld-ctrl'

    class Component(object):
        def check(self):
            assert self.value == 'ham'
```

In the previous example, this means that the 'field' component would be decorated with that extra `check()` method.

Rules are:

1. the special class must be called *Component*
2. that class should derive from *object* (but `ComponentProxy` is also allowed)
3. the component will still be a *ComponentProxy* , will *never* really inherit that special *Component* class
4. methods of the *Component* class will be bound to the *ComponentProxy*, hence `self` will refer to the *ComponentProxy* instance
5. *Component* may also have properties, static methods
6. private members are **not** copied to the *ComponentProxy*. They are discarded.
7. that applies to `__init__()` too. Components are volatile, they have no ordinary object lifecycle

8. if `MyScope` were to define a *Component* class, this would *not* apply to the "fld" component
9. but, if 'fld-ctrl' class is overriden, and the overriding class also defines a *Component* subclass, then *both* will have their members decorating "fld" component.

CHAPTER 4

Magic filters

Locating a particular row in some long list (table) of data would be very inefficient with manners. Reason is, the lazy way manners build/discover sub-components and attributes.

Consider the following snippet:

```
table = context.cur_page['section']['story']['haystack']

for row in table['rows']:
    if row['col_4'].text == 'needle':
        needle_row = row
        break
else:
    raise CAssertionError("Cannot find needle in haystack",
                          component=table)
```

Manners up to v0.11 would need to iterate over all rows, and for each one of those compute all columns (up to 'col_4'), then go back and retrieve the 'text' attribute of that element, to compare with 'needle'. Each of those steps would involve 1-3 selenium commands to achieve. It would take a few seconds (depending on the position of that row) to reach there.

Magic filters can provide a much faster solution to this case.

Assuming that code can be refactored as:

```
table = context.cur_page['section']['story']['haystack']

for row in table['rows'].filter(lambda r: r['col_4'].text == 'needle'):
    needle_row = row
    break
else:
    raise CAssertionError("Cannot find needle in haystack",
                          component=table)
```

then this loop could be resolved in as little as *one* Selenium call. See, magic filters operate on that *lambda* function and can reduce it to an XPath locator, before rows of the table would be iterated over. Thus, the expensive part is pushed from Python (manners) up to the browser (XPath) to locate and match.

Not all operators are currently supported in that lambda. It /could/ even be a named function or object method. But can only refer to sub-items using *[]*, attributes and equality checks. This is work in progress.

Note:: in situations like the above example, if 'rows' were to be called like *row_%d* after their position, *.filter()* would never get that number right, since it locates the desired one by skipping any non-matching ones (but cannot count the skipped ones).

CHAPTER 5

manners API: modules

5.1 behave_manners package

5.1.1 Subpackages

behave_manners.pagelems package

These classes represent a *map* of the site/page to be scanned/tested

They are the /definition/ describing interesting parts of the target page, not the elements of that page.

Submodules

behave_manners.pagelems.base_parsers module

class BaseDPOParser(*root_element*)

Bases: html.parser.HTMLParser, object

close()

Handle any buffered data.

get_result()

handle_charref(*name*)

handle_comment(*data*)

Comments are ignored

handle_data(*data*)

Reject any non-whitespace text in elements

handle_decl(*decl*)

handle_endtag(*tag*)

handle_entityref(*name*)

handle_pi(*data*)

handle_starttag(*tag*, *attrs*)

logger = None

unknown_decl(*data*)

class DBaseLinkElement(*tag*, *attrs*)
Bases: *behave_manners.pagelems.base_parsers.DPageElement*

Baseclass for <*link*> elements in any pagelem document

Links have mandatory "rel", "href" attributes and optional "url", "title".

consume(*element*)

is_empty = True

reduce(*site=None*)
Cleanup internally, possibly merging nested elements

None can be returned, in case this element is no longer useful. *reduce()* shall be called *after* all children have been scanned and appended to this; after thet have been reduced.

If *site* is provided, this should be a context object, which provides (or consumes) additional resources from this element.

class DOMScope(*parent*, *templates=None*)
Bases: `object`

A simple holder of shared components or variables across DOM levels

class Component
Bases: `object`

clear()
Clears the text if it's a text entry element.

click()
Clicks the element.

get_attribute(*name*)
Gets the given attribute or property of the element.

This method will first try to return the value of a property with the given name. If a property with that name doesn't exist, it returns the value of the attribute with the same name. If there's no attribute with that name, `None` is returned.

Values which are considered truthy, that is equals "true" or "false", are returned as booleans. All other non-`None` values are returned as strings. For attributes or properties which do not exist, `None` is returned.

Args

- name - Name of the attribute/property to retrieve.

Example:

```
# Check if the "active" CSS class is applied to an element.
is_active = "active" in target_element.get_attribute("class")
```

get_property(*name*)
Gets the given property of the element.

Args

- name - Name of the property to retrieve.

Usage

```
text_length = target_element.get_property("text_length")
```

is_displayed()
Whether the element is visible to a user.

is_enabled()
Returns whether the element is enabled.

is_selected()
Returns whether the element is selected.

Can be used to check if a checkbox or radio button is selected.

send_keys(**value*)
Simulates typing into the element.

Args

- value - A string for typing, or setting form fields. For setting file inputs, this could be a local file path.

Use this to send simple key events or to fill out form fields:

```
form_textfield = driver.find_element_by_name('username')
form_textfield.send_keys("admin")
```

This can also be used to set file inputs.

```
file_input = driver.find_element_by_name('profilePic')
file_input.send_keys("path/to/profilepic.gif")
# Generally it's better to wrap the file path in one of the methods
# in os.path to return the actual path to support cross OS testing.
# file_input.send_keys(os.path.abspath("path/to/profilepic.gif"))
```

submit()
Submits a form.

child()
Return a child scope linked to this one

get_template(*key*)

root_component

take_component(*comp*)

timeouts = {}

wait_js_conditions = []

class DPageElement(*tag=None*, *attrs=()*)
Bases: object

Base class for elements scanned in pagetemplate html

This is an abstract class, only subclasses shall be instantiated.

consume(*element*)

is_empty = False

iter_attrs(*webelem=None*, *scope=None*, *xpath_prefix=''*)
Iterate names of possible attributes

returns iterator of (name, descriptor)

iter_items(*remote*, *scope*, *xpath_prefix=''*, *match=None*)
Iterate possible children components

Parameters

- **remote** – remote WebElement to work on
- **scope** – DOMScope under which to operate
- **xpath_prefix** – hanging xpath from parent element
- **match** – if provided, only look for those items currently only a string is expected in *match*

Returns tuple (name, welem, ptmpl, scope)

pretty_dom()
Walk this template, generate (indent, name, xpath) sets of each node

Used for debugging, pretty-printing of parsed structure

reduce(*site=None*)
Cleanup internally, possibly merging nested elements

None can be returned, in case this element is no longer useful. *reduce()* shall be called *after* all children have been scanned and appended to this; after thet have been reduced.

If *site* is provided, this should be a context object, which provides (or consumes) additional resources from this element.

tag = ''

xpath

xpath_locator(*score*, *top=False*)
Return xpath locator of this and any child elements

Parameters

- **score** – mutable Integer, decrement depending on precision of locators
- **top** – locate exactly this (the top) element, or else any of its children

Returns str

class DataElement(*data*)
Bases: *behave_manners.pagelems.base_parsers.DPageElement*

append(*other*)

consume(*element*)

is_empty = True

set_full(*full*)

xpath_locator(*score*, *top=False*)
Return xpath locator of this and any child elements

Parameters

- **score** – mutable Integer, decrement depending on precision of locators
- **top** – locate exactly this (the top) element, or else any of its children

Returns str

exception HTMLParseError(*msg*, *position=(None*, *None)*)
Bases: Exception

Exception raised for all parse errors.

behave_manners.pagelems.dom_components module

Proxies of selenium WebElements into abstract structure of page's information

Goal of the PageTemplates/Component is to reduce the multi-level DOM structure of a web-page to a compact structure of /Components/ , which then convey the semantic information of that DOM and are easy to assert in testing. Components may be trivial, corresponding to single DOM elements, or not, being abstractions over complex DOM structures. They /should/ have their entry point mapped to a particular DOM element.

Components must have a well-defined /path/ to be addressed with. This cannot be guaranteed by this framework, but rather should be achieved through careful design of the page templates. Heavily dependant on the web-page's structure and technology. A good path is one that would map the "same" DOM element to a specific path, through subsequent readings of the webpage. Even if the webpage's DOM has been re-built by the JS framework of the page (React, Angular, etc.) .

Example: a table. Using a *<pe-repeat>* template element, rows of that table could be mapped to components. For static tables, path could just be the row number of those rows. But this becomes non-deterministic for tables that are, say, grids with dynamic sorting or infinite scroll. On those, key to the rows should be some primary key of the row data, like the remote database ID or unique name of that row-data.

Components should be considered volatile. By design, this framewor does NOT hold a reference of children components to one, but rather re-maps children on demand. Likewise, values of attributes shall always be fetched from the remote, each time the Component attribute is read. No caching. It is the caller's responsibility to copy the Component attributes to some other variable, if caching (rather than multiple WebDriver requests) is desired.

Unreachable components should be handled graceously. They would still raise an exception all the way up, but plugins may help in debugging, like by highlighting the visual position in the webpage where something is missing.

class CSSProxy(*parent*)
Bases: object

class ComponentProxy(*name*, *parent*, *pagetmpl*, *webelem*, *scope*)
Bases: behave_manners.pagelems.dom_components._SomeProxy

Cross-breed of a Selenium element and DPO page object

component_name

filter(*clause*, *safe=True*)
Iterator over sub-components that satisfy a condition

Usage:

```
for row in table.filter(lambda r: r['col_4'].text == "bingo!"):
    print("Found the bingo row:", row)
```

equivalent to:

```
for row in table.values():
    if row['col_4'].text == "bingo!":
        print("Found the bingo row:", row)
```

clause must be a function, which evaluates against a component and returns True whenever that component should participate in the result.

IFF *clause* is simple enough, *filter()* may optimize it to resolve the iteration in a very efficient search.

filter_gen(*clause*, *safe=True*)
Generator of *.filter()* functions

Just because the optimization in *.filter()* may be expensive to compute, it can be done once (offline) and re-applied multiple times to generate many iterator instances

path

class PageProxy(*pagetmpl*, *webdriver*, *scope*)
Bases: behave_manners.pagelems.dom_components._SomeProxy

Root of Components, the webpage

Holds reference to remote WebDriver, has no parent

path

behave_manners.pagelems.dom_meta module

class DOM_Meta
Bases: f3utils.service_meta._ServiceMeta

Metaclass for *DOMScope* ones.

Prepares a dictionary of descriptors that the scope can apply to components or pages under it. Operates at class initialization phase,

behave_manners.pagelems.exceptions module

exception CAssertionError(*arg*, *component=None*)
Bases: AssertionError, *behave_manners.pagelems.exceptions.ComponentException*

exception CAttributeError(*arg*, *msg=None*, *component=None*)
Bases: AttributeError, *behave_manners.pagelems.exceptions.ComponentException*

exception CAttributeNoElementError(*arg*, *msg=None*, *component=None*)
Bases: *behave_manners.pagelems.exceptions.CAttributeError*

Raised when component attribute cannot be retrieved because of missing element

Special case of *CAttributeError*, may need to be handled explicitly. Implies *NoSuchElementException* from remote DOM.

exception CKeyError(*arg*, *msg=None*, *component=None*)
Bases: KeyError, *behave_manners.pagelems.exceptions.ComponentException*

exception CValueError(*arg*, *component=None*)
Bases: ValueError, *behave_manners.pagelems.exceptions.ComponentException*

class ComponentException(*component=None*, *msg=None*)
Bases: object

Mixin for exceptions that can refer to faulty component

The component would be an instance of *ComponentProxy*

exception ElementNotFound(*msg=None*, *screen=None*, *stacktrace=None*, *parent=None*, *selector=None*, *pagelem_name=None*)
Bases: selenium.common.exceptions.NoSuchElementException

Raised when a *WebElement* cannot locate a child, refers to that parent

pretty_parent
String of parent element, in pretty format

exception PageNotReady
Bases: AssertionError

Raised when browser has not finished rendering/settled the page

exception Timeout
Bases: AssertionError

Raised when any action or element fails to respond in time

exception UnwantedElement(*msg=None*, *screen=None*, *stacktrace=None*)
Bases: selenium.common.exceptions.NoSuchElementException

behave_manners.pagelems.helpers module

class Integer(*value*)
Bases: object

Mutable integer implementation

Useful for counters, that need to be passed by reference

class XPath(*xpath*, *complete=True*)
Bases: object

Dummy class to wrap an xpath string

Attribute complete mark if this XPath should reliably locate elements or have less reliable results.

count_calls(*fn*)
Wrapper for function, keeping a count of fn's calls

prepend_xpath(*pre*, *xpath*, *glue=False*)
Prepend some xpath to another, properly joining the slashes

textescape(*tstr*)

to_bool(*v*)
Convert boolean-like value of html attribute to python True/False

Example truthy values (for *attr* in <b>): <b attr> <b attr="1"> <b attr="true"> <b attr="anything">

example falsy values: <b attr="0"> <b attr="false"> <b attr="False">

behave_manners.pagelems.index_elems module

class IHeadObject(*tag=None*, *attrs={}*)
Bases: *behave_manners.pagelems.index_elems.ISomeObject*

reduce(*site=None*)
Cleanup internally, possibly merging nested elements

None can be returned, in case this element is no longer useful. *reduce()* shall be called *after* all children have been scanned and appended to this; after thet have been reduced.

If *site* is provided, this should be a context object, which provides (or consumes) additional resources from this element.

class IHtmlObject(*tag=None*, *attrs=()*)
Bases: *behave_manners.pagelems.index_elems.ISomeObject*

Consume the <html> element as top-level index page

class ILinkObject(*tag*, *attrs*)
Bases: *behave_manners.pagelems.base_parsers.DBaseLinkElement*

class ISomeObject(*tag=None*, *attrs=()*)
Bases: *behave_manners.pagelems.base_parsers.DPageElement*

Base class for 'index.html' and its sub-elements

These override *reduce()* so that <link> elements are consumed and the rest is discarded.

reduce(*site=None*)
Cleanup internally, possibly merging nested elements

None can be returned, in case this element is no longer useful. *reduce()* shall be called *after* all children have been scanned and appended to this; after thet have been reduced.

If *site* is provided, this should be a context object, which provides (or consumes) additional resources from this element.

class IndexHTMLParser(*site_collection*)
Bases: *behave_manners.pagelems.base_parsers.BaseDPOParser*

Parser only for 'index.html', containing links to other pages

In this pseydo-HTML site language, 'index.html' is only allowed to contain `<link>` elements to other named page object files. This parser restricts any other HTML elements for this file.

Example:

```
<html>
    <link rel="next" href="main-page.html" title="Main Page" url="/">
    <link rel="preload" href="common-components.html">
</html>
```

handle_starttag(*tag*, *attrs*)

logger = <Logger behave_manners.pagelems.index_elems.IndexHTMLParser (WARNING)>

behave_manners.pagelems.loaders module

class BaseLoader
Bases: object

Abstract base for loading DPO html files off some storage

Subclass this to enable loading from any kind of storage.

multi_open(*filepattern*, *directory=''*, *mode='rb'*)
Open a set of files (by glob pattern) for reading

Returns iterator of open files

open(*fname*, *directory=''*, *mode='rb'*)
Open file at *fname* path for reading.

Return a context manager file object

class FSLoader(*root_dir*)
Bases: *behave_manners.pagelems.loaders.BaseLoader*

Trivial filesystem-based loader of files

multi_open(*filepattern*, *mode='rb'*)
Open a set of files (by glob pattern) for reading

Returns iterator of open files

open(*fname*, *mode='rb'*)
Open file at *fname* path for reading.

Return a context manager file object

behave_manners.pagelems.main module

cmdline_main()
when sun as a script, this behaves like a syntax checker for DPO files

behave_manners.pagelems.page_elements module

class DomContainerElement(*tag=None*, *attrs=()*)
Base class for 'regular' DOM elements that can contain others

reduce(*site=None*)
Cleanup internally, possibly merging nested elements

None can be returned, in case this element is no longer useful. *reduce()* shall be called *after* all children have been scanned and appended to this; after thet have been reduced.

If *site* is provided, this should be a context object, which provides (or consumes) additional resources from this element.

class LeafElement(*tag*, *attrs*)
Generic element, that has no sub-elements

class NamedElement(*tag*, *attrs*)
Generic element, defining DOM component through 'this' attribute

pretty_dom()
Walk this template, generate (indent, name, xpath) sets of each node

class Text2AttrElement(*name*, *strip=False*)
Internal pagelem node that retrieves text as an attribute to DOM component

This is instantiated when template has eg. * [text] * . The text is not asserted, but rather appointed to an attribute of parent component.

consume_after(*element*)
Turn this into a partial text matcher, after some element tag

consume_before(*element*)
Turn this into a partial text matcher, before some tag

behave_manners.pagelems.scopes module

class Angular5App (*parent*, *templates=None*)
Bases: *behave_manners.pagelems.scopes.WaitScope*

Scope of an application using Angular 5+

wait_js_conditions = ["if (document.readyState != 'complete') { return 'document'; }",

class AngularJSApp (*parent*, *templates=None*)
Bases: *behave_manners.pagelems.scopes.WaitScope*

Scope of an application using AngularJS (1.x)

wait_js_conditions = ["if (document.readyState != 'complete') { return 'document'; }",

class Fresh (*comp*)
Bases: object

Keep a component fresh, recovering any stale children under it

class GenericPageScope (*parent*, *templates=None*)
Bases: *behave_manners.pagelems.scopes.WaitScope*

Default page scope

Page scope is the one attached to the remote DOM 'page', ie the <html> element. A good place to define page-wide properties, such as waiting methods.

Wait for JS at least

class RootDOMScope (*templates=None*, *site_config=None*)
Bases: *behave_manners.pagelems.base_parsers.DOMScope*

Default scope to be used as parent of all scopes

Such one would be the parent of a 'page' scope.

Given that scopes can use the attributes of parent ones, this is where global attributes (such as site-wide config) can be set.

component_class
alias of *behave_manners.pagelems.dom_components.ComponentProxy*

class WaitScope (*parent*, *templates=None*)
Bases: *behave_manners.pagelems.base_parsers.DOMScope*

class Page
Bases: object

wait_all (*timeout*)

isready_all (*driver*)
Signal that all actions are settled, page is ready to contine

Override this to add any custom logic besides *isready_js()* condition.

isready_js (*driver*)
One-off check that JS is settled

Parameters **driver** – WebDriver instance

resolve_timeout (*timeout*)

timeouts = {'long': 60.0, 'medium': 10.0, 'short': 2.0}

wait(*timeout='short'*, *ready_fn=None*, *welem=None*, *webdriver=None*)
Waits until 'ready_fn()' signals completion, times out otherwise

wait_all(*timeout='short'*, *welem=None*, *webdriver=None*)
Waits for all conditions of *isready_all()*

wait_js_conditions = ["if (document.readyState != 'complete') { return 'document'; }",

behave_manners.pagelems.site_collection module

class DSiteCollection(*loader*, *config=None*)
Bases: *behave_manners.pagelems.base_parsers.DPageElement*

Collection of several HTML pages, like a site

Not supposed to be parsed, but directly instantiated as root hierarchy of all parsed html pageobject files.

consume(*element*)

get_by_file(*fname*)
Get page by template filename

Returns pageelem

get_by_title(*title*)
Get page by set title

Titles are arbitrary, pretty names assigned to page templates :return: (pageelem, url)

get_by_url(*url*, *fragment=None*)
Find the page template that matches url (path) of browser

returns (page, title, params)

get_root_scope()
Return new DOMScope bound to self

A *DSiteCollection* can be reused across runs, but ideally each run should start with a new scope, as returned by this method. Then, this scope should be passed to pages under this site, as those returned by *get_by_title()* , *get_by_url()* etc.

load_all()
Load all referenced pages

Used for forced scan of their content

load_galleryfile(*pname*)

load_index(*pname*)

load_pagefile(*pname*)

load_preloads()
Load pending preloads or gallery files

logger = <Logger site_collection (WARNING)>

register_link(*link*)

behave_manners.steplib package

Submodules

5.1.2 Submodules

behave_manners.action_chains module

Wrapper around *selenium.common.action_chains* that uses domComponents as inputs

class ActionChains(*component*)
Bases: object

Mirror of *selenium.common.action_chains* for domComponents

drag_and_drop(*source*, *target*)

behave_manners.context module

class GContext(***kwargs*)
Bases: object

new(***kwargs*)
Return context manager object, new level down current context

Usage:

```
my_context.a = 0
with my_context.new(a=1) as ctx:
    ctx.b = 2
    assert my_context.a == 1
    assert my_context.b == 2
assert my_context.a == 0
```

pop()

push()

behave_manners.dpo_run_browser module

Standalone runner of 'DPO' structures, validate against an externally loaded Selenium WebDriver instance.

cmdline_main()
when sun as a script, this behaves like a syntax checker for DPO files

behave_manners.dpo_validator module

Standalone runner of 'DPO' structures, validate against an externally loaded Selenium WebDriver instance.

class ExistingRemote(*command_executor*, *session_id*, *saved_capabilities*, *desired_capabilities={}*, *saved_w3c=None*, ***kwargs*)
Bases: selenium.webdriver.remote.webdriver.WebDriver

Remote webdriver that attaches to existing session

start_session(*desired_capabilities*, *browser_profile=None*)
Creates a new session with the desired capabilities.

Args

- browser_name - The name of the browser to request.
- version - Which browser version to request.
- platform - Which platform to request the browser on.
- javascript_enabled - Whether the new session should support JavaScript.
- browser_profile - A selenium.webdriver.firefox.firefox_profile.FirefoxProfile object. Only used if Firefox is requested.

cmdline_main()
when sun as a script, this behaves like a syntax checker for DPO files

import_step_modules(*paths*, *modules*)
Import any python module under 'paths', store its names in 'modules/xx'

This is used to implicitly register ServiceMeta classes defined in these step modules.

shorten_txt(*txt*, *maxlen*)

behave_manners.screenshots module

Utilities for browser screenshots, connected to events

class Camera(*base_dir='.'*)
Bases: object

A camera takes screenshots (or element shots) of the browser view

An instance will be attached to the behave context. Configured by site configuration. Then triggered each time a step explicitly wants a screenshot, or the hooks implicitly catching some failure.

capture_missing_elem(*context*, *parent*, *missing_path*)
Screenshot of browser when some element is missing

Parameters

- **context** – behave Context containing site and browser
- **parent** – selenium.WebElement under which other was not found
- **missing_path** – string of XPath missing

highlight_element(*context*, *component=None*, *webelem=None*, *color=None*, *border=None*)
Perform some action with an element visually highlighted

This should place a square rectangle on the DOM, around the element in question. The rectangle is appended into the root of the DOM to avoid any inheritance effects of the interesting element. After the action is performed, the highlight element is removed.

highlight_js = "\n var highlight = document.createElement('div');\n highlight.setAttri

snap_failure(*context*, **args*)

snap_success(*context*, **args*)

take_shot(*context*, *mode=''*)
Capture the full browser viewport.

behave_manners.site module

class ChromeWebContext(*context*, *config=None*)
Bases: *behave_manners.site.WebContext*

Web context for Chromium browser

class FakeContext
Bases: object

Dummy *behave* context

Will substitute a proper *behave* context for when manners stuff is run outside of a *behave* test suite.

class Config
Bases: object

setup_logging()

userdata = {}

class Runner
Bases: object

add_cleanup(*fn*, **args*)

close()

log = <Logger context (WARNING)>

class FirefoxWebContext(*context*, *config=None*)
Bases: *behave_manners.site.WebContext*

Web context for Chromium browser

class GenericWebContext(*context*, *config=None*)
Bases: *behave_manners.site.WebContext*

class IExploderWebContext(*context*, *config=None*)
Bases: *behave_manners.site.WebContext*

Web context for Chromium browser

exception RemoteNetworkError(*message*, ***kwargs*)
Bases: *behave_manners.site.RemoteSiteError*

exception RemoteSiteError(*message*, ***kwargs*)
Bases: Exception

Raised on errors detected at remote (browser) side

class SiteContext(*context*, *config=None*)
Bases: object

Holds (web)site information in a behave context

A SiteContext is attached to behave's context like *context.site = SiteContext(. . .)*

and from there on tests can refer to site-wide attributes through that *context.site* object.

base_url

init_collection(*loader=None*)

class WebContext(*context*, *config=None*)
Bases: *behave_manners.site.SiteContext*

Site context when a browser needs to be launched

fourOfours = ('/favicon.ico',)

get_cur_title(*context*)
Return pretty title of page currently loaded on the browser

launch_browser(*context*)
Launch a browser, attach it to *context.browser*

navigate_by_title(*context*, *title*, ***kwargs*)
Open a URL, by pretty title

navigate_by_url(*context*, *url*, ***kwargs*)

process_logs(*context*, *consumer=None*)
Fetch logs from browser and process them

Parameters **silent** – suppress exceptions arising from logs

update_cur_page(*context*)
Update *context.cur_page* when URL may have changed

validate_cur_page(*context*, *max_depth=10000*)
Validates current browser page against pagelem template

Current page will be checked by url and the page template will be traversed all the way down.

current_context()
Discover current behave context from caller stack

Use this inside an object that has no reference to behave context, but need to use it. Note that the context may be deliberately omitted, but in some exceptional cases would be useful to have.

behave_manners.site_setup module

site_setup(*context*, *config=None*, *extra_conf=None*, *loader=None*)
Load a config file and prepare site for operation

Follow the config for site, browser and page elements settings, store them in `behave.Context` .

behave_manners.step_utils module

implies(*step_text*)

CHAPTER 6

Tutorial 1: Get started

A short tutorial on how to get started (from scratch) using *behave_manners*.

Note: The need to keep this tutorial simple and small contradicts with the core purpose of *manners*. Manners are designed for the complex case, for large sites, while the examples below are over-simplified. Please, keep that in mind. Gains from *manners* come when the site is much harder than this example.

6.1 1. Project setup

A minimal project for behave_manners should have the following files:

environment.py Standard *behave* script for setting up configuration defaults

config.yaml Recommended configuration file with project defaults, base settings.

requirements.txt Recommended to have a file listing all python dependencies used for this project. Should contain a line for *behave-manners*

site/index.html Required, mapping of URLs to template files & titles

site/first-page.html At least one template file (to cover one or multiple site URLs)

first-page.feature At least one feature file with scenarios

steps/first_page_steps.py At least one python script with implementations of the feature steps

Having a *setup.py* is not really needed: think of the tests project as a collection of data files; they should be runable from a simple checkout.

Behave-manners should only need minimal content in the above files; tries to fill-in reasonable defaults for most un-mentioned values.

6.1.1 Examples of project files

environment.py

```
from behave_manners import site_setup


def before_all(context):
    site_setup(context, config='config.yaml')
```

This is the only line needed for behave-manners to set-up everything: the browser, load the page templates (lazily) and set the base-url for all your site.

config.yaml

```
site:
    base_url: https://material.angular.io

browser:
    engine: chrome
    launch_on: feature

    screenshots:
        dir: screenshots

page_objects:
    index: site/index.html
```

As names suggest, contains settings for the site, browser tuning and points to the page templates that should be used.

requirements.txt

```
behave-manners
```

site/index.html

```
<html>
<head>
    <link rel="next" href="first-page.html" title="Home Page" url="/">
    <link rel="next" href="quickstart.html" title="Quick Start" url="/guide/getting-
↪started">
</head>
</html>
```

index.html is a special file. Can only have a <*head*> containing <*link*> elements, to establish the mapping of URLs to the page templates and optionally titles. It is only a table; could have been in any other format, yet done in html for symmetry with the rest of page templates.

site/first-page.html

```
<html>
<body>
<material-docs-app ng-version="[ng-version]">
    <div pe-deep class="docs-header-start">
        <a this="get-started-button">
            <span>Get started</span>
        </a>
        <pe-not>
```

(continues on next page)

(continued from previous page)

```
            <div class="bad-section">Foo
            </div>
        </pe-not>
    </div>
</material-docs-app>
</body>
</html>
```

Each page template file should match the remote DOM of the site being tested. See documentation for further explanation of the page template format.

first-page.feature

```
Feature: Check home page

Scenario: Use the search form

  Given I am at the "Home Page"
   When I click get-started-button
   Then I am directed to "Quick Start"
```

Feature files describe the desired tests in an abstract, high-level way.

steps/first_page_steps.py

```
from behave import given, when, then, step


@given(u'I am at the "{page}"')
def step_impl1(context, page):
    context.site.navigate_by_title(context, page)
    context.cur_element = context.cur_page

@when(u'I click {button}')
def click_a_button(self, button):
    context.cur_element[button].click()

@then(u'I am directed to "{page}"')
def check_this_page(self, page):
    title = context.site.update_cur_page(context)
    assert title == page, "Currently at %s (%s)" % (title, context.browser.current_
↪url)
```

Python implementations of steps is the 'glue' between features and the abstract Component tree that *behave-manners* can provide. Here, page elements (and nested sub-elements of) can be referenced like simple Python objects, also interacted with.

6.2 2. Python setup

Assuming that python is installed and operational, it is highly recommended that your project uses a dedicated virtual environment.

Within that virtualenv, only need to install `behave-manners` . Or, even better, call:

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

to cover any other dependencies your project may desire.

6.2.1 Chromedriver

The browser driver (chromedriver, here) needs to be installed separately, as a binary, into your system.

Calling `which chromedriver` within the virtualenv should verify if it is properly placed (and executable).

6.3 3. Verifying page templates

After the *index.html* and page templates are written, they can be tested independently of feature files (and step definitions). For this, *behave-manners* provides with a pair of utilities:

- behave-run-browser
- behave-validate-remote

Which are complementary: 'run-browser' will launch a browser with the settings as specified in 'config.yaml' . Then 'validate-remote' can be called repeatedly against that browser, to scan the page on that browser and print the Components that are discoverable in it.

Example from the above settings, in 'https://material.angular.io'

```
$ behave-run-browser .
INFO:main:Entering main phase, waiting for browser to close
INFO:main:Browser changed to: Angular Material
...

$ behave-validate-remote
INFO:site_collection:Read index from 'site/index.html'
INFO:site_collection:Read page from 'site/first-page.html'
INFO:main:Got page First Page ()

    <Page "https://material.angular.io/">
      get-started-button <a class="docs-button mat-raised-button">

INFO:main:Validation finished, no errors
```

The standard output of the second is the Page component, and within it, that 'get-started' button. Real-life examples should be much more deep than that.

CHAPTER 7

Tutorial 2: writing page elements

Step-by-step demonstration on how page elements work and get related to the target DOM.

Note: The example below includes code snippets from MDN site, by Mozilla Contributors and licensed under CC-BY-SA 2.5.

Note: MDN site's HTML is too good for *manners*. It is well written, clean and structured. Manners can do much worse than that.

7.1 HTML vs HTML

Open MDN in the div element . Open the inspector to examine the DOM of that page.

The body of the page, contracted, looks like this:

```
<body data-slug="Web/HTML/Element/div" contextmenu="edit-history-menu" data-search-
↪url="" class="document">

<script>... </script>
    <ul id="nav-access">... </ul>

    <header id="main-header" class="header-main">...</header>

    <main id="content" role="main">...</main>
    <footer id="nav-footer" class="nav-footer">...</footer>
</body>
```

And, focusing a part of the `<main>` element we could see a snippet like:

```
<div id="toc" class="toc">
    <div class="center">
    <div class="toc-head">Jump to:</div>
        <ol class="toc-links">
        <li><a href="#Attributes" rel="internal" class="">Attributes</a></li>
        <li><a href="#Usage_notes" rel="internal" class="">Usage notes</a></li>
        <li><a href="#Examples" rel="internal" class="">Examples</a></li>
        <li><a href="#Specifications" rel="internal" class="">Specifications</a></li>
        <li><a href="#Browser_compatibility" rel="internal" class="">Browser␣
↪compatibility</a></li>
        <li><a href="#See_also" rel="internal" class="toc-current">See also</a>
        </li></ol>
    </div>
</div>
```

In principle, pasting the above section into a pagelem template file, should work. It would match 1:1 to the MDN site DOM, that page.

7.1.1 Removing redundant elements

Not all html elements are worth matching. Say, that `<div class="center">` needs not be that specific, nor the "Jump to:" text is any more specific than the `id="toc"` of the outer `<div>`

So, the above snippet could be cleaned like:

```
<div id="toc">
    <div>
    <ol class="toc-links">
        <li><a href="#Attributes">Attributes</a></li>
        <li><a href="#Usage_notes">Usage notes</a></li>
        <li><a href="#Examples">Examples</a></li>
        <li><a href="#Specifications">Specifications</a></li>
        <li><a href="#Browser_compatibility">Browser compatibility</a></li>
        <li><a href="#See_also" class="toc-current">See also</a>
        </li>
    </ol>
    </div>
</div>
```

7.1.2 Adding components

The above will not *emit* any matched components, until `this` attributes are set in the interesting elements:

```
<div id="toc" this="page-toc">
    <div>
    <ol class="toc-links" this="links">
        <li this="Attributes"><a href="#Attributes">Attributes</a></li>
        <li><a href="#Usage_notes">Usage notes</a></li>
        <li><a href="#Examples">Examples</a></li>
        <li this="Specifications"><a href="#Specifications">Specifications</a></li>
        <li><a href="#Browser_compatibility">Browser compatibility</a></li>
        <li><a href="#See_also" class="toc-current">See also</a>
        </li>
    </ol>
```

(continues on next page)

(continued from previous page)

```
    </div>
</div>
```

That would produce a structure of components like:

```
[page-toc]
  - [links]
      - [Attributes]
      - [Specifications]
```

7.1.3 Generalizing

Then, the indidual `<li>` elements should be covered all with one rule, rather than hard-coding their exact attributes. This allows the same *toc* code to match any table of contents that conforms to this layout.

Pagelem code now can be simplified like:

```
<div id="toc" this="page-toc">
    <div>
    <ol class="toc-links" this="links">
        <li this="[title]"><a href="[href]">[title]</a></li>
    </ol>
    </div>
</div>
```

Getting simpler, and will also match all entries in the TOC this way.

`href="[href]"` syntax means that the value of `href=` attribute will be assigned to a `href` attribute of the resulting component. Will match anything in there. Likewise `[title]` as text of an element will copy whatever text into a `title` attribute.

Then, `this="[title]"` means that the value assigned to `title` becomes the name of the resulting component.

Thus, the resulting component structure after this generalisation should now be:

```
[page-toc]
  - [links]
      - [Attributes]
          href = #Attributes
          title = Attributes
      - [Usage notes]
          href = #Usage_notes
          title = Usage notes
      - [Examples]
          href = #Examples
          title = Examples
      ...
```

7.2 Full page

The above example cannot work standalone; rather it needs to be put in context of a full HTML page. Assuming the earlier structure, it would need to be written as:

```
<html>
<body>
    <main id="content" role="main">
        <div id="toc" this="page-toc">
            <div>
            <ol class="toc-links" this="links">
                <li this="[title]"><a href="[href]">[title]</a></li>
            </ol>
            </div>
        </div>
    </main>
</body>
</html>
```

This works for the MDN case, because “toc” is just one level down from “main”. But would become worse if “toc” had been somewhere deep within the page DOM.

In that case, intermediate levels could be skipped with:

```
<html>
<body>
        <div pe-deep id="toc" this="page-toc">
            <ol pe-deep class="toc-links" this="links">
                <li this="[title]"><a href="[href]">[title]</a></li>
            </ol>
        </div>
</body>
</html>
```

making code now more compact.

7.3 Templating

Since the TOC, footer and menu of this site are expected to be re-occuring across all pages, makes sense to write them as re-usable templates

gallery.html

```
<html>
<body>
    <template id="toc">
        <div pe-deep id="toc" this="page-toc">
            <ol pe-deep class="toc-links" this="links">
                <li this="[title]"><a href="[href]">[title]</a></li>
            </ol>
        </div>
    </template>

    <template id="header">
        <header id="main-header" class="header-main">...</header>
    </template>

    <template id="footer">
        <footer id="nav-footer" class="nav-footer">... </footer>
    </template>
```

(continues on next page)

(continued from previous page)

```
</body>
</html>
```

div-element.html

```
<html>
<head>
    <link rel="import" href="gallery.html">
</head>
<body>
    <use-template id="header"/>

    <main id="content" role="main">
        <use-template id="toc"/>

        <!--custom code for catching the div-element page content -->
    </main>

    <use-template id="footer"/>
</body>
</html>
```

Templates may be called repeatedly in some page, even recursively.

7.4 Attribute matching

Attributes of pagelems will match same attributes on remote DOM, by default.

Example:

```
<div class="toc">
```

will match a *div* only if it's class equals to "toc". That's not always convenient, since other classes may exist along "toc", that matching should ignore.

```
<div class="+toc">
```

will then match if the div's class *contains* "toc".

Then, several values could be or-ed by mentioning them each:

```
<div role="document" role="article">
```

Writing `title="[tooltip]"` will NOT attempt any match, but transfer the value of remote DOM attribute `title` to Component attribute `.tooltip` . That can co-exist with a matcher, like:

```
<div class="[class_]" class="+important">
```

Note here the underscore after `class_` , because attribute names need to be valid Python variable names, `class` is a reserved word.

Attribute matching can be negated with the `!` operator:

```
<div role="!contentinfo">
```

or, when the element should not contain a class:

```
<div class="!+hidden">
```

7.5 Other pagelems

The *pagelem* HTML-like language offers some other extra elements and attributes that help match remote DOM with less code on the testing side.

Components may be tagged as `pe-optional` rather than failing the match. Pagelem can match regardless of DOM tag with `<pe-any>` element.

Alternate structures may be matched with `<pe-choice>` . Within a repetition or `<pe-choice>` , collections of elements can be forced to go together in a `<pe-group>`.

More about them can be read in the reference and supplied examples.

CHAPTER 8

Indices and tables

8.1 Built-in page elements Reference

8.1.1 Standard HTML elements

Some of the standard HTML elements need special handling, therefore they have corresponding classes defined in pagelems.

Otherwise, any other elements will be matched verbatim

class DHtmlObject(*tag: <html>*)
Consume the <html> element as top-level site page

As expected, may only have a <*head*> and a <*body*> .

class DHeadElement(*tag: <head>*)
HTML head only supports parsing-related elements

Currently, only <link>s and <template>s are allowed in <head>

class DLinkObject(*tag: <link>*)
Links are used to include other pagelem templates

class DBodyElement(*tag: <body>*)
Defines, matches the HTML <body>

class ScriptElement(*tag: <script>*)
Scripts are NOT allowed so far

class InputElement(*tag: <input>*)
Model an <input> element

Inputs are special and a bit weird. They are points of interaction with the remote side, also volatile, so must be exposed into DOM components. When *this* is specified, inputs become components as usual. Otherwise *they become attributes* of the parent components.

Therefore <*input*> elements MUST have *this*, an *id* or a *name*.

When 'type' is specified at this pagelem node, it helps choose the right descriptor class for the value of this input. If not, it may be auto-detected IF *this* is used OR *name='*'* performs wildcard detection of input elements.

If 'pe-name' attribute is specified, it overrides the 'name' one as the name to be used under the component, but does NOT need to match remote attribute(s).

Example:

```
<input pe-name="submit" type="submit">
```

will match any submit button and assign it to 'submit' name, even if the remote <*input*> element has no more attributes.

Note that setting both 'this' and 'pe-name' makes no sense, since a component-ized <*input*> element will not need a name.

class TextAreaObj(*tag: <textarea>*)
Textarea is handled just like <input>

class GHtmlObject(*tag: <html> * in gallery*)
Handle the <html> element of a gallery file

Gallery files can only contain <*template*> elements in their <head> or <body>, no other HTML.

8.1.2 Templates and slots

Templates and slots are defined in HTML5. This concept is re-used in pagelems, but as a means of parsing. See: *Templates and Slots*

class DTemplateElement(*tag: <template>*)
A template defines reusable DOM that is not normally rendered/scanned

See: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/template

Likewise, the template will be re-used in this parser, match remote DOM and generate same proxy elements, under their caller.

id="name"
Mandatory attribute, needed to register this template in the local scope.

class DUseTemplateElem(*tag: <use-template>*)
Calls a <template> in its place

Example:

```
<use-template id="chapter">
    <div slot="extra" class="remark" this="remark"> ... </div>
</use-template>
```

id="name"
refers to a <*template*> to use

class DSlotElement(*tag: <slot>*)
The contents of a <slot> are replaced by parent scope, if available

This resembles the official W3C definition of slots in the browser, meant to be a placeholder for content that can be customized within a template.

name="slot-name"
Slots must have a name, so that implementations can attach to them.

class DSlotContentElement (*tag: <pe-slotcontent>*)
Jump back to the content of calling '<slot>' element

Example:

```
<div class="slot">
    <slot name="foo">
        <div class="content">
        </div>
    </slot>
</div>

<div slot="foo" class="bar">
    <middle>
        <pe-slotcontent/>
    </middle>
</div>
```

Should be equivalent to:

```
<div class="slot">
    <div class="bar">
        <middle>
            <div class="content"></div>
        </middle>
    </div>
</div>
```

This is inspired by Jinja2 'caller' concept: http://jinja.pocoo.org/docs/2.10/templates/#call

8.1.3 Pagelem flow elements

class DeepContainObj (*tag: <pe-deep>*)
Match contained elements at any level deep inside the DOM

Example:

```
<body>
    <pe-deep> <div class="content"></div> </pe-deep>
</body>
```

will match:

```
<body>
    <main>
        <div class="section">
            <div class="content"> ... </div>
        </div>
    </main>
</body>
```

Equivalent of using 'pe-deep' as an attribute.

Example (alt):

```
<body>
    <div pe-deep class="content"></div>
</body>
```

class RootAgainElem(*tag: <pe-root>*)
Reset to root element (of DOM), keep component deep in tree

Useful for overlays, where the modal part is attached to the <body> instead of that element itself, but logically linked to that inner component.

class RepeatObj(*tag: <pe-repeat>*)
Locate multiple components, from one template

In its simpler form, it allows explicit iterations, where the inner component wouldn't.

Example:

```
<pe-repeat>
    <div class="chapter" this="chapter> ...</div>
</pe-repeat>
```

this would match *all * , assigning them to components like *chapter1* , *chapter2* etc.

But also useful for any controlled iteration. Supports `min` and `max` limits of how many components to match.

Then, <*pe-repeat*> can have `this` attribute, assigning the list of produced components as one component. This is useful even when iteration is *not* desired, rather a component that does not *consume* its corresponding DOM (parent) element.

class PeChoiceElement(*tag: <pe-choice>*)
Matches the first child of this element (at least)

Using *pe-choice* means that at least one of its children shall match the remote, any others can fail.

Example:

```
<pe-choice>
    <div class="chapter" this="chapter">...</div>
    <div class="footer" this="footer">...</div>
    <section class="index" this="index"> ... </section>
</pe-choice>
```

the above would succeed if any of these three patterns match, but could also emit /all/ of them, if these can be found.

Note that the output of <*pe-choice*> is ordered by the options given inside *pe-choice*, NOT the order that elements are in the remote DOM. This is due to the XPath implementation.

class PeGroupElement(*tag: <pe-group>*)
Trivial group, DOM-less container of many elements

Implements all-or-nothing logic, will never produce partial components* from its childen.

By default, unordered behaviour. Can specify 'ordered' to enforce that children sub-elements are matched in strict order.

* unless the group's children have optional logic with <*pe-choice*> or <*pe-repeat*>.

class PeMatchIDElement(*tag: <pe-matchid>*)
Jump to an element that matches by id

It is a compact shorthand for taking an element's attribute, resolving it and then searching the root of the DOM for that *id*.

Example:

```
<input this="input" role="combobox"
       aria-owns='[owns]'>
<pe-matchid id="root['input'].owns" this="panel" pe-optional>
    ...
</pe-matchid>
```

would resolve the *aria-owns* attribute of the <*input*> , then look for any element with that *id* in the tree, assign it to the dropdown panel.

8.1.4 Pagelem match elements

class AnyElement(*tag: <pe-any>*)
Match any HTML element

This would match any element in the remote DOM, but also serves as a baseclass for matching particular tags.

Useful for matching a particular element by attribute, rather than HTML tag.

Offers some standard attributes and rich syntax for matching remote DOM element properties.

this="name"
Exposes the element as a component under that name

slot="name"
Attaches this element into a <*slot*> of a <*template*>

Only valid within `<use-template>` *blocks*

pe-deep
Matches that element at any nested level under the parent. Equivalent of puting that element within a <*pe-deep*> tag

pe-optional
Makes matching optional, this element and any children may not exist.

pe-controller="some.controller"
Uses *some.controller* to complement functionality of this component.

pe-ctrl
Synonym of `pe-controller`

class PeNotElement(*tag: <pe-not>*)
Negative-match element: check that element does NOT contain others

Negative-match will invert the meaning of contained matches, thus not selecting parents that contain specified patterns.

Example:

```
<div class="eggs">
    <pe-not><div class="spam"></div>
    </pe-not>
</div>
```

Meaning that it will match a *div@class=eggs* that does NOT contain a spam *div* .

When multiple children elements are specified inside pe-not, then *all* of them should be present for parent to mis-match. Having any of the children but not all, will allow the parent to match.

The other logic, failing the parent if any children exist, is possible by using multiple <*pe-not*> elements.

Do not use named elements (with *this*) or logic of *pe-choice*, *pe-repeat* or pe-optional elements inside a *pe-not*. As it will never create components, such logic is pointless.

class RegexElement (*tag: <pe-regex>*)
Match text of remote DOM element, parse with regex into attribute(s)

If this regex contains `(?P<name>...)` named groups, these will be exposed as *name* attributes.

Otherwise, if *this* attribute is defined, expose *all* matched string under this name.

Note: text inside this element can contain '<' and '>', no need to escape these.

Example:

```
<div class="header" this="chapter">
    <pe-regex>Chapter (?P<number>[0-9]+): (?P<title>.*)</pe-regex>
</div>
```

This one would match DOM like this:

```
<div class="header">Chapter 4: Getting there</div>
```

and produce a component like:

```
chapter: number="4" title="Getting there"
```

8.1.5 Pagelem data elements

class PeDataElement (*tag: <pe-data>*)
Define arbitrary data as a component attribute

This supports two modes: with a *value=* attribute or using inner JSON data.

When using the *value=* attribute, the data will be a plain string. When using inner JSON, it can be any of the simple types that JSON supports.

Example:

```
<div class="chapter" this="chapter">
    <pe-data name="lang" value="en"/>
    <pe-data name="difficulty">0.8</pe-data>
    ...
</div>
```

would produce a component like: chapter: lang="en" difficulty=0.8

Note that *difficulty* is parsed as JSON, and therefore becomes a native float, rather than a string.

class PEScopeDataElement (*tag: <pe-scopedata>*)
Attach arbitrary data as a scope attribute

WARNING: this data will only work if the component containing this tag is 'discovered', ie. attached to that scope.

- genindex
- modindex

- search

Python Module Index

b

Index

D

E

F

G

H

I

L

M

N

O

P