Build Test Cases Using VTF

The '*runtime*' dictionary

Tuple structure for the key-value pair: 'keyName': (<action>, <locator> [, <value>])

Style note: all dictionary keys use camel case, e.g. 'firstName'

The <action> items

- Click – click on an object
  - ('Click', '//*[@id="licenseGrid_form"]/a')
  - ('Click', '//*[@id="licenseGrid_form"]/a', '')
- Type – type or input text into a text field or textarea
  ('Type', '#date_granted', '05042015')
- Select – select an item from a dropdown, select by visible text
  ('Select', '#state_code_id', 'California')
  Note: 'California' represents the text that appears in the dropdown
  May also use this format which does the same as above, but allows you to use a different select types, i.e. text, id, or value.
  ('Select', '#state_code_id', {'value': '&state;', 'select_type': 'text'})
  ('Select', '#state_code_id', {'value': '&state;'})
- Wait – wait for an element to appear in the DOM
  ('Wait', '#date_granted', {'wait_time': 8, 'condition': 'element_to_be_clickable'})
  The 'Wait' is a special case that uses a dictionary as the <value> portion; uses two keys, a wait_time key that overrides the default of 5 seconds, and a condition key that is a string type and provides the automation tester the ability to specify the expected condition name, i.e. presence_of_element_located, element_to_be_selected, etc.—see http://selenium-python.readthedocs.io/waits.html#explicit-waits
- Chain – link several actions together—see below for more details

The <locator> structure

- Locator's xpath starts with '//', e.g. '**//***[@id="ribbon_form"]/ul/li[1]/div[1]'
- Locator's class starts with '.', e.g. '**.**charlie-sierra'
- Locator's id starts with '#', e.g. '**#**name_id'
- Locator's tag starts with '<', e.g. '**<**tag_name>'
- Locator's css starts with 'css=', e.g. '**css=**a.leaf.btname_id'
- Locator's name starts with 'name=', e.g. '**name=**name_id'

Note: Special case '<tag_name>' looks for many elements; helps get around the display of items that have dynamic id attribute; e.g. id="client_5568b9eecbc2e". The assumption is that the last tag is the desired element. Another way around this is to use the css as the web element locator.

The <value> types – string, number, or dictionary

A Sample - Create an action sequence in a low level base class

Situation: Need to navigate to a common location, i.e. “Licensing landing page” that several test cases for licensing may need to use.

Solution: Create a low level base class that uses this action sequence.

```
from ui import UI
class License(UI):
   def __init__(self, override=None):
       super().__init__()
       runtime = {
           'license': ('Click', '//*[@id="slide-out"]/li[3]/ul/li/a/i',),
           'landing': ('Click', '//*[@id="slide-out"]/li[3]/ul/li/div/ul/li[1]/a'),
       }
       process = UI(override)
       process.update(runtime)
       order = ('license', 'landing', )
       process.execute(order)}
```

Description: This action sequence illustrates the ‘Click’ action. The ‘runtime’ variable holds the actions and those actions are assigned when the process.update(runtime) is invoked. The ‘order’ variable contains the sequence of execution. The process.execute(order) executes the actions. Not all runtime elements may be a part of the runtime sequence—see the next sample.

Reason: The xpath for the “license” and “landing” elements are

- License -> '//*[@id="slide-out"]/li[3]/ul/li/a/i'
- Landing -> '//*[@id="slide-out"]/li[3]/ul/li/div/ul/li[1]/a'

Dev adds two new menu elements *before* the ‘license’ element, which push the license element down the list to the fifth place. We change the element’s xpath to be '//*[@id="slide-out"]/li[5]/ul/li/{etc.}', so all test cases that use this base class are unaffected.

Remove an existing action from a base class

Situation: The base class has an action you do not want your test case to perform.

Solution: Override the action's key within the derived class with the value of **None**.

Example: The base class Checklist has runtime data set as…

```
runtime = {

    'showAll': ('Click', '#checklist-form-container'),

    'provider': ('Click', "#license-grid1', ),

}
```

You do not want to click on the element for 'showAll', so you'll need to override the base class element 'showAll' in the derived class—the test case—as follows:

```
from ui import UI

from ui.low.license import License

from ui.high.checklist import Checklist

Class AddCarrier(UI):

       License()

       override = {'showAll': None }  # ← set the override HERE

       Checklist(override)  # Checklist({'showAll': None})

       etc.
```

Override a key inside the base class – the '*override*' dictionary

Situation: The base class has a key whose 'value' you want to override

Solution: Override the action's key with the new string value

Example: The base class has runtime data

```
runtime = {

    'selectType': ('Select', '#education_type_id', 'Medical School')

    'provider': ('Click', "#license-grid1', ),

}
```

You want to select 'Doctorate Degree' instead of 'Medical School' for locator 'selectType', so you'll need to override it inside the test case as follows:

```
override = {'selectType': 'Doctorate Degree', }

Base(override)
```

The runtime variable looks like this after the override is finished:

```
runtime = {

    'selectType': ('Select', '#education_type_id', 'Doctorate Degree')

    'provider': ('Click', "#license-grid1', ),

}
```

Override an action's element inside the base class

Situation: The base class has an xpath, and/or value you want to override

Solution: Override the action's key inside the derived class with the new xpath and/or value

Example: The base class has runtime data that has an xpath—`#education_type_id`—that needs to be changed.

```
runtime = {

    'selectType': ('Select', '#education_type_id', 'Medical School')

    'provider': ('Click', "#license-grid1', ),

}
```

You want to select a different element with a different value. We create an override inside the test case:

```
override = {'selectType': ('Select', '#education_degree_id', 'Doctorate Degree'), }

Base(override)
```

Use a placeholder to replace a common hardcoded index

Situation: You want the test case to select a different row inside the same element, but the xpath has a hardcoded index value

Solution: Embed a placeholder to take the place of the xpath's index value and use a second key that holds the placeholder's default index value.

Example: Placeholders can be inserted inside an locator's xpath. For example, an xpath that comes from a table can have several rows,

The original xpath: `//*[@id="license_grid"]/tbody/tr[1]/td[8]/a`

The modified xpath: `//*[@id="license_grid"]/tbody/tr[&rowNum;]/td[8]/a`

The placeholder **&rowNum;** allows any derived class to override the default. The placeholder must have a runtime key by the same name. The base class—this example uses Checklist—has the runtime dict that looks like this:

```
runtime = {
        'rowNum': '1', # first row
        'provider': ('Click', //*[@id="license_grid"]/tbody/tr[&rowNum;]/td[8]/a')
}
```

Note: The placeholder key is not part of the 'order' tuple.

A derived class wants to use a different row, so it does this by passing in an override:

```
override = {'rowNum': '3', }  # third row
Checklist(override)
```

Example: A combination of the above concepts

```
runtime = {
        'selectType': ('Select', '#education_type_id', 'Medical School'),
        'placeholder': '1',
        'provider': ('Click', "#license-grid&placeholder;', ),
}
```

A derived class wants to override the placeholder and the selection option:

```
override = {
        'selectType': 'Doctorate Degree',
        'placeholder': '2'}
Checklist(override)
```

The resulting runtime after override will look like this:

```
runtime = {
        'selectType': ('Select', '#education_type_id', 'Doctorate Degree'),
        'provider': ('Click', "#license-grid2', ),
}
```

Alternate Select action by using a different select type

Situation: The standard select type ‘text’ is too slow, so want to use a ‘value’ or ‘index’ type

Solution: Use a dictionary inside the runtime element’s <value> slot

Example: Three select types are available, ‘value’, ‘text’, or ‘index’. The ‘text’ select type is the default. These types access selenium’s Select methods:

```
select.select_by_index(index)
select.select_by_visible_text(text)
select.select_by_value(value)
```

The following sample shows <value> dictionary using placeholders, i.e. ‘state’ and ‘type’.

```
runtime = {
    'state': 'UT',
    'type': 'Home',
    'addrState': ('Select', "#state', {'value': '&state;', 'select_type': 'value'}),
    'addrType': ('Select', "#type', {'value': '&type;', 'select_type': 'index'}),
}
```

You may use ‘select_type’: ‘text’ as well. This example shows the tuple without the placeholder.

```
runtime = {
    'addrType': ('Select', "#type', {'value': 'Other'}),
}
```

```
runtime = {
    'addrType': ('Select', "#type', {'value': 'Other', 'select_type': 'text'}),
}
```

Both above samples work the same, but since ‘text’ is the default, the select_type key may be omitted, as shown in the first sample.

The following sample is the standard select type version:

```
runtime = {
    'addrType': ('Select', "#type', 'Other'),
}
```

A Sample - Create an Action Chain

Situation: Need to chain several commands together

Solution: Create a chain of action sequences.

```
runtime = {  # from ui.low.license
    'level': '5',
    'license': ("Chain", [
        ('move_to_element', {
            'to_element':
                '//*[@id="slide-out"]/li[&level;]/ul/li/a/i[1]'}),
        ('click', {
            'on_element':
                '//*[@id="slide-out"]/li[&level;]/ul/li/a/i[1]'}),
        ('click', {
            'on_element':
                '//*[@id="slide-out"]/li[&level;]/ul/li/div/ul/li[1]/a'}
         ),
    ]),
}
```

Description: This chain of actions are stored in an ActionChain object – see [Selenium Site]

Reason: The application will sometimes hang and fail to dive into hidden elements, i.e. Nav bar click License | License Landing Page fails intermittently. The ActionChain helps resolve this.