ftw.testbrowser

Release 2.1.2

Jul 28, 2020

Contents

ftw.testbrowser is a browser library for testing Plone based web sites and applications.

CHAPTER 1

Introduction

ftw.testbrowser is a browser library for testing Plone based web sites and applications (CI).

1.1 Features

The test browser supports all the basic features:

- Visit pages of the Plone site
- Access page content
- Find nodes by CSS- and XPath-Expressions or by text
- Click on links
- Fill and submit forms
- File uploading
- Make WebDAV requests

The *ftw.testbrowser* also comes with some basic Plone page objects.

ftw.testbrowser currently does not support JavaScript.

1.2 Motivation

A test browser should have a simple but powerful API (CSS expressions), it should be fast, reliable and easy to setup and use.

The existing test browsers for Plone development were not satisfactory:

- The zope.testbrowser, which is the current standard for Plone testing does not support CSS- or XPath-Selectors, it is very limiting in form filling (buttons without names are not selectable, for example) and it leads to brittle tests.
- The splinter test browser has a zope driver and various selenium based drivers. This abstraction improves the API but it is still limiting since it bases on *zope.testbrowser*.
- The robotframework is a selenium based full-stack browser which comes with an own language and requires a huge setup. The use of selenium makes it slow and brittle and a new language needs to be learned.

There are also some more browser libraries and wrappers, usually around selenium, which often requires to open a port and make actual requests. This behavior is very time consuming and should not be done unless really necessary, which is usally for visual things (making screenshots) and JavaScript testing.

1.3 How it works

The *ftw.testbrowser* uses mechanize with plone.testing configurations / patches to directly dispatch requests in Zope.

The responses are parsed in an lxml.html document, which allows us to do all the necessary things such as selecting HTML elements or filling forms.

While querying, *ftw.testbrowser* wraps all the HTML elements into node wrappers which extend the *lxml* functionality with things such as using *CSS* selectors directly, clicking on links or filling forms based on labels.

CHAPTER 2

Quickstart

Add *ftw.testbrowser* to your testing dependencies in your *setup.py*:

```
tests_require = [
    'ftw.testbrowser',
    ]

setup(name='my.package',
      install_requires=['Plone'],
      tests_require=tests_require,
      extras_require=dict(tests=tests_require))
```

Write tests using the browser:

```
from ftw.testbrowser import browsing
from ftw.testbrowser.pages import factoriesmenu
from ftw.testbrowser.pages import plone
from ftw.testbrowser.pages import statusmessages
from plone.app.testing import PLONE_FUNCTIONAL_TESTING
from plone.app.testing import SITE_OWNER_NAME
from unittest2 import TestCase


class TestFolders(TestCase):

    layer = PLONE_FUNCTIONAL_TESTING

    @browsing
    def test_add_folder(self, browser):
        browser.login(SITE_OWNER_NAME).open()
        factoriesmenu.add('Folder')
        browser.fill({'Title': 'The Folder'}).submit()

        statusmessages.assert_no_error_messages()
        self.assertEquals('folder_listing', plone.view())
        self.assertEquals('The Folder', plone.first_heading())
```

CHAPTER 3

User documentation

3.1 Setup

For using the test browser, just decorate your test methods with the *@browsing* decorator.

```
from ftw.testbrowser import browsing
from unittest2 import TestCase
from plone.app.testing import PLONE_FUNCTIONAL_TESTING


class TestMyView(TestCase):

    layer = PLONE_FUNCTIONAL_TESTING

    @browsing
    def test_view_displays_things(self, browser):
        browser.visit(view='my_view')
```

Warning: Make sure that you use a functional testing layer!

See also:

ftw.testbrowser.browsing()

By default there is only one, global browser, but it is also possible to instantiate a new browser and to set it up manually:

```
from ftw.testbrowser.core import Browser

browser = Browser()
app = zope_app

with browser(app):
    browser.open()
```

Warning: Page objects and forms usually use the global browser. Creating a new browser manually will not set it as global browser and page objects / forms will not be able to access it!

3.1.1 Choosing the default driver

The default driver is chosen automatically, depending on the Zope version and whether the browser is setup with a Zope app or not. Without a Zope app the default driver is `LIB_REQUESTS`. With Zope 4 the default is `LIB_WEBTEST` and with Zope 2 it's `LIB_MECHANIZE`.

`LIB_WEBTEST` is only available with Zope 4 (Plone 5.2 and later) while `LIB_MECHANIZE` and `LIB_TRAVERSAL` are only available with Zope 2.

The default driver can be changed on the browser instance, overriding the automatic driver selection:

```
from ftw.testbrowser.core import Browser
from ftw.testbrowser.core import LIB_MECHANIZE
from ftw.testbrowser.core import LIB_REQUESTS
from ftw.testbrowser.core import LIB_TRAVERSAL
from ftw.testbrowser.core import LIB_WEBTEST
```

(continues on next page)

(continued from previous page)

```
browser = Browser()
# always use mechanize:
browser.default_driver = LIB_MECHANIZE

# or always use webtest:
browser.default_driver = LIB_WEBTEST

# or always use requests:
browser.default_driver = LIB_REQUESTS

# or use traversal in the same transactions with same connection:
browser.default_driver = LIB_TRAVERSAL
```

When using the testbrowser in a `plone.testing` layer, the driver can be chosen by using a standard `plone.testing` fixture:

```
from ftw.testbrowser import MECHANIZE_BROWSER_FIXTURE
from ftw.testbrowser import REQUESTS_BROWSER_FIXTURE
from ftw.testbrowser import TRAVERSAL_BROWSER_FIXTURE
from ftw.testbrowser import WEBTEST_BROWSER_FIXTURE
from plone.app.testing import PLONE_FIXTURE
from plone.app.testing import FunctionalTesting


MY_FUNCTIONAL_TESTING_WITH_MECHANIZE = FunctionalTesting(
    bases=(PLONE_FIXTURE,
           MECHANIZE_BROWSER_FIXTURE),
    name='functional:mechanize')

MY_FUNCTIONAL_TESTING_WITH_REQUESTS = FunctionalTesting(
    bases=(PLONE_FIXTURE,
           REQUESTS_BROWSER_FIXTURE),
    name='functional:requests')

MY_FUNCTIONAL_TESTING_WITH_TRAVERSAL = FunctionalTesting(
    bases=(PLONE_FIXTURE,
           TRAVERSAL_BROWSER_FIXTURE),
    name='functional:traversal')

MY_FUNCTIONAL_TESTING_WITH_WEBTEST = FunctionalTesting(
    bases=(PLONE_FIXTURE,
           WEBTEST_BROWSER_FIXTURE),
    name='functional:webtest')
```

3.2 Visit pages

For visiting a page, use the *visit* or *open* method on the browser (those methods do the same).

Visiting the Plone site root:

```
browser.open()
print browser.url
```

See also:

ftw.testbrowser.core.Browser.url()

Visiting a full url:

```
browser.open('http://nohost/plone/sitemap')
```

Visiting an object:

```
folder = portal.get('the-folder')
browser.visit(folder)
```

Visit a view on an object:

```
folder = portal.get('the-folder')
browser.visit(folder, view='folder_contents')
```

The *open* method can also be used to make POST request:

```
browser.open('http://nohost/plone/login_form',
             {'__ac_name': TEST_USER_NAME,
              '__ac_password': TEST_USER_PASSWORD,
              'form.submitted': 1})
```

See also:

ftw.testbrowser.core.Browser.open()

3.3 Logging in

The *login* method sets the *Authorization* request header.

Login with the *plone.app.testing* default test user (*TEST_USER_NAME*):

```
browser.login().open()
```

Logging in with another user:

```
browser.login(username='john.doe', password='secret')
```

Logout and login a different user:

```
browser.login(username='john.doe', password='secret').open()
browser.logout()
browser.login().open()
```

See also:

ftw.testbrowser.core.Browser.login(), *ftw.testbrowser.core.Browser.logout()*

3.4 Finding elements

Elements can be found using CSS-Selectors (*css* method) or using XPath-Expressions (*xpath* method). A result set (*Nodes*) of all matches is returned.

See also:

ftw.testbrowser.nodes.Nodes()

CSS:

```
browser.open()
heading = browser.css('.documentFirstHeading').first
self.assertEquals('Plone Site', heading.normalized_text())
```

See also:

ftw.testbrowser.core.Browser.css(), ftw.testbrowser.nodes.NodeWrapper.normalized_text()

XPath:

```
browser.open()
heading = browser.xpath('h1').first
self.assertEquals('Plone Site', heading.normalized_text())
```

See also:

ftw.testbrowser.core.Browser.xpath()

Finding elements by text:

```
browser.open()
browser.find('Sitemap').click()
```

The *find* method will look for theese elements (in this order):

- a link with this text (normalized, including subelements' texts)
- a field which has a label with this text
- a button which has a label with this text

See also:

ftw.testbrowser.core.Browser.find()

3.5 Matching text content

In HTML, most elements can contain direct text but the elements can also contain sub-elements which also have text.

When having this HTML:

```
<a id="link">
    This is
    <b>a link
</a>
```

We can get only direct text of the link:

```
>>> browser.css('#link').first.text
'\n        This is\n        '
```

or the text recursively:

```
>>> browser.css('#link').first.text_content()
'\n        This is\n        a link\n    '
```

See also:

ftw.testbrowser.nodes.NodeWrapper.text_content()

or the normalized recursive text:

```
>>> browser.css('#link').first.normalized_text()
'This is a link'
```

See also:

ftw.testbrowser.nodes.NodeWrapper.normalized_text()

Functions such as *find* usually use the *normalized_text*.

See also:

ftw.testbrowser.core.Browser.find()

3.6 Get the page contents / json data

The page content of the currently loaded page is always available on the browser:

```
browser.open()
print browser.contents
```

See also:

ftw.testbrowser.core.Browser.contents()

If the result is a JSON string, you can access the JSON data (converted to python data structure already) with the *json* property:

```
browser.open(view='a-json-view')
print browser.json
```

See also:

ftw.testbrowser.core.Browser.json()

3.7 Filling and submitting forms

The browser's *fill* method helps to easily fill forms by label text without knowing the structure and details of the form:

```
browser.visit(view='login_form')
browser.fill({'Login Name': TEST_USER_NAME,
              'Password': TEST_USER_PASSWORD}).submit()
```

The *fill* method returns the browser instance which can be submitted with *submit*. The keys of the dict with the form data can be either field labels (<*label*> text) or the name of the field. Only one form can be filled at a time.

3.7.1 File uploading

For uploading a file you need to pass at least the file data (string or stream) and the filename to the *fill* method, optionally you can also declare a mime type.

There are two syntaxes which can be used.

Tuple syntax:

```
browser.fill({'File': ('Raw file data', 'file.txt', 'text/plain')})
```

Stream syntax

```
file_ = StringIO('Raw file data')
file_.filename = 'file.txt'
file_.content_type = 'text/plain'

browser.fill({'File': file_})
```

You can also pass in filesystem files directly, but you need to make sure that the file stream is opened untill the form is submitted.

```
with open('myfile.pdf') as file_:
    browser.fill({'File': file_}).submit()
```

See also:

ftw.testbrowser.core.Browser.fill(), *ftw.testbrowser.form.Form.submit()*, *ftw.testbrowser.form.Form.save()*

3.8 Tables

Tables are difficult to test without the right tools. For making the tests easy and readable, the table components provide helpers especially for easily extracting a table in a readable form.

For testing the content of this table:

```
<table id="shopping-cart">
    <thead>
        <tr>
            <th>Product</th>
            <th>Price</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Socks</td>
            <td>12.90</td>
        </tr>
        <tr>
            <td>Pants</td>
            <td>35.00</td>
        </tr>
    </tbody>
    <tfoot>
        <tr>
            <td>TOTAL:</td>
            <td>47.90</td>
        </tr>
    </tfoot>
</table>
```

You could use the `lists` method:

```
self.assertEquals(
    [['Product', 'Price'],
     ['Socks', '12.90'],
     ['Pants', '35.00'],
     ['TOTAL:', '47.90']],
    browser.css('#shopping-cart').first.lists())
```

See also:

ftw.testbrowser.table.Table.lists()

or the `dicts` method:

```
self.assertEquals(
    [{'Product': 'Socks',
      'Price': '12.90'},
     {'Product': 'Pants',
      'Price': '35.00'},
     {'Product': 'TOTAL:',
      'Price': '47.90'}],
    browser.css('#shopping-cart').first.dicts())
```

See also:

ftw.testbrowser.table.Table.dicts()

See the tables API for more details.

See also:

ftw.testbrowser.table.Table(), *ftw.testbrowser.table.TableRow()*, *ftw.testbrowser.table.TableCell()*

3.9 Page objects

ftw.testbrowser ships some basic page objects for Plone. Page objects represent a page or a part of a page and provide an API to this part. This allows us to write simpler and more expressive tests and makes the tests less brittle.

Read the post by Martin Fowler for better explenation about what page objects are.

You can and should write your own page objects for your views and pages.

See the API documentation for the page objects included in *ftw.testbrowser*:

- The **plone** page object provides general information about this page, such as if the user is logged in or the view / portal type of the page.
- The **factoriesmenu** page object helps to add new content through the browser or to test the addable types.
- The **statusmessages** page object helps to assert the current status messages.
- The **dexterity** page object provides helpers related to dexterity
- The **z3cform** page object provides helpers related to z3cforms, e.g. for asserting validation errors in the form.

See also:

ftw.testbrowser.pages

3.10 XML Support

When the response mimetype is `text/xml` or `application/xml`, the response body is parsed as XML instead of HTML.

This can lead to problems when having XML-Documents with a default namespace, because lxml only supports XPath 1, which does not support default namespaces.

You can either solve the problem yourself by parsing the `browser.contents` or you may switch back to HTML parsing. HTML parsing will modify your document though, it will insert a `html` node for example.

Re-parsing with another parser:

```
browser.webdav(view='something.xml')   # XML document
browser.parse_as_html()                # HTML document
browser.parse_as_xml()                 # XML document
```

See also:

ftw.testbrowser.core.Browser.parse_as_html

See also:

ftw.testbrowser.core.Browser.parse_as_xml

See also:

ftw.testbrowser.core.Browser.parse

3.11 HTTP requests

ftw.testbrowser also supports not following redirects. This is useful for testing the bodies of redirect responses or inspecting *Location* headers.

This is currently not implemented for mechanize.

```
from ftw.testbrowser import browsing
from unittest2 import TestCase


class TestRedirects(TestCase):

  @browsing
  def test_redirects_are_followed_automatically(self, browser):
      browser.open(view='test-redirect-to-portal')
      self.assertEquals(self.portal.absolute_url(), browser.url)
      self.assertEquals(('listing_view', 'plone-site'), plone.view_and_portal_type())

  @browsing
  def test_redirect_following_can_be_prevented(self, browser):
      browser.allow_redirects = False
      browser.open(view='test-redirect-to-portal')
      self.assertEquals('/'.join((self.portal.absolute_url(), 'test-redirect-to-portal
↪')), browser.url)
      self.assertEquals((None, None), plone.view_and_portal_type())
```

3.12 WebDAV requests

ftw.testbrowser supports doing WebDAV requests, although it requires a ZServer to be running because of limitations in mechanize.

Use a testing layer which bases on `plone.app.testing.PLONE_ZSERVER`:

```
from plone.app.testing import FunctionalTesting
from plone.app.testing import PLONE_FIXTURE
from plone.app.testing import PLONE_ZSERVER
from plone.app.testing import PloneSandboxLayer


class MyPackageLayer(PloneSandboxLayer):

    defaultBases = (PLONE_FIXTURE, )

MY_PACKAGE_FIXTURE = MyPackageLayer()
MY_PACKAGE_ZSERVER_TESTING = FunctionalTesting(
    bases=(MY_PACKAGE_FIXTURE,
           PLONE_ZSERVER),
    name='my.package:functional:zserver')
```

Then use the `webdav` method for making requests in the test:

```
from ftw.testbrowser import browsing
from my.package.testing import MY_PACKAGE_ZSERVER_TESTING
from unittest2 import TestCase


class TestWebdav(TestCase):

    layer = MY_PACKAGE_ZSERVER_TESTING

    @browsing
    def test_DAV_option(self, browser):
        browser.webdav('OPTIONS')
        self.assertEquals('1,2', browser.response.headers.get('DAV'))
```

See also:

ftw.testbrowser.core.Browser.webdav()

3.13 Error handling

The testbrowser raises exceptions by default when a request was not successful. When the response has a status code of *4xx*, a *ftw.testbrowser.exceptions.HTTPClientError* is raised, when the status code is *5xx*, a *ftw.testbrowser.exceptions.HTTPServerError* is raised.

When the requests is sent to a Plone CMS and causes an "insufficient privileges" result, a *ftw.testbrowser.exceptions.InsufficientPrivileges* is raised. The exception is raised for anonymous users (rendering the login form) as well as for logged in users (rendering the "Insufficient Privileges" page).

3.13.1 Disabling HTTP exceptions

Disable the `raise_http_errors` flag when the test browser should not raise any HTTP exceptions:

```
@browsing
def test(self, browser):
    browser.raise_http_errors = False
    browser.open(view='not-existing')
```

3.13.2 Expecting HTTP exceptions

Sometimes we want to make sure that the server responds with a certain bad status. For making that easy, the testbrowser provides assertion context managers:

```
@browsing
def test(self, browser):
    with browser.expect_http_error():
        browser.open(view='failing')

    with browser.expect_http_error(code=404):
        browser.open(view='not-existing')

    with browser.expect_http_error(reason='Bad Request'):
        browser.open(view='get-record-by-id')
```

3.13.3 Expecting unauthoirzed exceptions (Plone)

When a user is not logged in and is not authorized to access a resource, Plone will redirect the user to the login form (`require_login`). The `expect_unauthorized` context manager knows how Plone behaves and provides an easy interface so that the developer does not need to handle it.

```
@browsing
def test(self, browser):
    with browser.expect_unauthorized():
        browser.open(view='plone_control_panel')
```

3.13.4 Exception bubbling

Exceptions happening in views can not be catched in the browser by default. When using an internally dispatched driver such as Mechanize, the option `exception_bubbling` makes the Zope Publisher and Mechanize let the exceptions bubble up into the test method, so that it can be catched and asserted there.

```
@browsing
def test(self, browser):
    browser.exception_bubbling = True
    with self.assertRaises(ValueError) as cm:
        browser.open(view='failing')

    self.assertEquals('No valid value was submitted', str(cm.exception))
```

3.13.5 Plone 5: resource registries disabled

In Plone 5, the resource registries are cooked when the resource registry viewlets are rendered. Cooking the bundles takes a lot of time. Since `ftw.testbrowser` does nothing with JavaScript or CSS, cooking of resources is disabled by default for performance improvement. That means that `<script>` and `<styles>` tags are missing in the HTML. This can make the tests up to 4-5 times faster.

Tests or projects which require to have the resource tags in the HTML can reenable them.

Enable by browser flag:

```
@browsing
def test(self, browser):
    browser.disable_resource_registries = False
    browser.open()
```

Enable by environment variable:

```
TESTBROWSER_DISABLE_RESOURCE_REGISTRIES = false
```

CHAPTER 4

API Documentation

4.1 Browser

ftw.testbrowser.**MECHANIZE_BROWSER_FIXTURE = <Layer 'ftw.testbrowser.drivers.layers.Default**
A plone.testing layer which sets the default driver to Mechanize.

ftw.testbrowser.**REQUESTS_BROWSER_FIXTURE = <Layer 'ftw.testbrowser.drivers.layers.DefaultDr**
A plone.testing layer which sets the default driver to Requests.

ftw.testbrowser.**TRAVERSAL_BROWSER_FIXTURE = <Layer 'ftw.testbrowser.drivers.layers.Default**
A plone.testing layer which sets the default driver to Traversal.

ftw.testbrowser.**WEBTEST_BROWSER_FIXTURE = <Layer 'ftw.testbrowser.drivers.layers.DefaultDri**
A plone.testing layer which sets the default driver to Webtest.

ftw.testbrowser.**browser = <ftw.browser.core.Browser instance>**
The singleton browser instance acting as default browser.

ftw.testbrowser.**browsing**(*func*)
The `browsing` decorator is used in tests for automatically setting up the browser and passing it into the test function as additional argument:

```
from ftw.testbrowser import browsing
from plone.app.testing import PLONE_FUNCTIONAL_TESTING
from unittest import TestCase
```

(continues on next page)

(continued from previous page)

```
class TestSomething(TestCase):
    layer = PLONE_FUNCTIONAL_TESTING

    @browsing
    def test_login_form(self, browser):
        browser.open(view='login_form')
        self.assertEqual('http://nohost/plone/login_form',
                         browser.url)
```

class ftw.testbrowser.core.**Browser**

Bases: object

The Browser is the top level object of ftw.testbrowser. It represents the browser instance and is used for navigating and interacting with the browser.

The Browser is a context manager, requiring the Zope app to be set:

```
# "app" is the Zope app object

from ftw.testbrowser import Browser

browser = Browser()

with browser(app):
    browser.open()
```

When using the browser in tests there is a @browsing test-method decorator uses the global (singleton) browser and sets it up / tears it down using the context manager syntax. See the *ftw.testbrowser.browsing* documentation for more information.

Variables

- **raise_http_errors** – HTTPError exceptions are raised on 4xx and 5xx response codes when enabled (Default: True).
- **exception_bubbling** – When enabled, exceptions from within the Zope view are bubbled up into the test method if the driver supports it. (Default: False).

append_request_header(*name*, *value*)

Add a new permanent request header which is sent with every request until it is cleared.

HTTP allows multiple request headers with the same name. Therefore this method does not replace existing names. Use *replace_request_header* for replacing headers.

Be aware that the requests library does not support multiple headers with the same name, therefore it is always a replace for the requests module.

Parameters

- **name** (*string*) – Name of the request header
- **value** (*string*) – Value of the request header

See also:

replace_request_header()

See also:

clear_request_header()

base_url
The base URL of the current page. The base URL can be defined in HTML using a `<base>`-tag. If no `<base>`-tag is found, the page URL is used.

body
The binary response content

clear_request_header(*name*)
Removes a permanent header. If there are no such headers, the removal is silently skipped.

Parameters **name** (*string*) – Name of the request header as positional arguments

click_on(***kwargs*)
Find a link by its text and click on it.

Parameters

- **text** (*string*) – The text to be looked for.
- **within** (*ftw.testbrowser.nodes.NodeWrapper*.) – A node object for limiting the scope of the search.

Returns The browser object.

Raises *ftw.testbrowser.exceptions.NoElementFound*

See also:

find()

clone()
Creates a new browser instance with a cloned state of the current browser. Headers and cookies are copied but not shared. The new browser needs to be used as a context manager, eg.:

with browser.clone() as sub_browser: sub_browser.open()

Returns A new browser instance.

Return type *ftw.testbrowser.core.Browser*

contents
The response body as native string.

contenttype
The contenttype of the response, e.g. `text/html; charset=utf-8`.

See also:

mimetype(), *encoding()*

context
Returns the current context (Plone object) of the currently viewed page.

Returns The Plone context object

cookies
A read-only dict of current cookies.

css(***kwargs*)
Select one or more HTML nodes by using a *CSS* selector.

Parameters **css_selector** (*string*) – The CSS selector.

Returns Object containg matches.

Return type *ftw.testbrowser.nodes.Nodes*

debug()
Open the current page in your real browser by writing the contents into a temporary file and opening it with os.system `open [FILE]`.

This is meant to be used in pdb, not in actual code.

encoding
The encoding of the respone, e.g. `utf-8`.

See also:

contenttype()

expect_http_error(***kwds*)
Context manager for expecting certain HTTP errors. The `code` and `reason` arguments may be provided or omitted. The values are only asserted if the arguments are provided. An assertion error is raised when the HTTP error is not cathed in the code block. The code block may make a request or reload the browser.

Parameters

- **code** (`int`) – The status code to assert.
- **reason** (`string`) – The status reason to assert.

Raises `AssertionError`

expect_unauthorized(***kwds*)
Context manager for expecting that next request, issued in the context manager block, will be unauthorized.

fill(*values*)
Fill multiple fields of a form on the current page. All fields must be in the same form.

Example:

```
browser.open(view='login_form')
browser.fill({'Login Name': 'hugo.boss', 'Password': 'secret'})
```

Since the form node (*ftw.testbrowser.form.Form*) is returned, it can easily be submitted:

```
browser.open(view='login_form')
browser.fill({'Login Name': 'hugo.boss',
              'Password': 'secret'}).submit()
```

Parameters **values** (*dict*) – The key is the label or input-name and the value is the value to set.

Returns The form node.

Return type *ftw.testbrowser.form.Form*

find(*text*, *within=None*)
Find an element by text. This will look for:

- a link with this text (normalized, including subelements' texts)
- a field which has a label with this text
- a button which has a label with this text

Parameters

- **text** (*string*) – The text to be looked for.

- **within** (*ftw.testbrowser.nodes.NodeWrapper*.) – A node object for limiting the scope of the search.

Returns A single node object or *None* if nothing matches.

Return type *ftw.testbrowser.nodes.NodeWrapper*

find_button_by_label(*label*, *within=None*)
Finds a form button by its text label.

Parameters

- **text** (*string*) – The text to be looked for.
- **within** (*ftw.testbrowser.nodes.NodeWrapper*.) – A node object for limiting the scope of the search.

Returns The button node or *None* if nothing matches.

Return type *ftw.testbrowser.form.SubmitButton*

find_field_by_text(*text*, *within=None*)
Finds a form field which has *text* as label.

Parameters

- **text** (*string*) – The text to be looked for.
- **within** (*ftw.testbrowser.nodes.NodeWrapper*.) – A node object for limiting the scope of the search.

Returns A single node object or *None* if nothing matches.

Return type *ftw.testbrowser.nodes.NodeWrapper*

find_form_by_field(*field_label_or_name*)
Searches for a field and returns the form containing the field. The field is searched by label text or field name. If no field was found, *None* is returned.

Parameters **label_or_name** (*string*) – The label or the name of the field.

Returns The form instance which has the searched fields or *None*

Return type *ftw.testbrowser.form.Form*.

find_form_by_fields(**labels_or_names*)
Searches for the form which has fields for the labels passed as arguments and returns the form node.

Returns The form instance which has the searched fields.

Return type *ftw.testbrowser.form.Form*

Raises *ftw.testbrowser.exceptions.FormFieldNotFound*

Raises *ftw.testbrowser.exceptions.AmbiguousFormFields*

find_link_by_text(*text*, *within=None*)
Searches for a link with the passed text. The comparison is done with normalized whitespace and includes the full text within the link, including its subelements' texts.

Parameters

- **text** (*string*) – The text to be looked for.
- **within** (*ftw.testbrowser.nodes.NodeWrapper*.) – A node object for limiting the scope of the search.

Returns The link object or *None* if nothing matches.

Return type *ftw.testbrowser.nodes.LinkNode*

form_field_labels
A list of label texts and field names of each field in any form on the current page.

The list contains the whitespace normalized label text of the each field. If there is no label or it has an empty text, the fieldname is used instead.

Returns A list of label texts (and field names).

Return type list of strings

forms
A *dict* of form instance where the key is the *id* or the *name* of the form and the value is the form node.

get_driver(*library=None*)
Return the driver instance for a library.

headers
A dict of response headers.

json
If the current page is JSON only, this can be used for getting the converted JSON data as python data structure.

login(*username='test-user'*, *password='secret'*)
Login a user by setting the `Authorization` header.

logout()
Logout the current user by removing the `Authorization` header.

mimetype
The mimetype of the respone, e.g. `text/html`.

See also:

contenttype()

on(*url_or_object=None*, *data=None*, *view=None*, *library=None*)
`on` does almost the same thing as `open`. The difference is that `on` does not reload the page if the current page is the same as the requested one.

Be aware that filled form field values may stay when the page is not reloaded.

See also:

open()

open(*url_or_object=None*, *data=None*, *view=None*, *library=None*, *referer=False*, *method=None*, *headers=None*, *send_authenticator=False*)
Opens a page in the browser.

Request library: When running tests on a Plone testing layer and using the `@browsing` decorator, the `mechanize` library is used by default, dispatching the request internal directly into Zope. When the testbrowser is used differently (no decorator nor zope app setup), the `requests` library is used, doing actual requests. If the default does not fit your needs you can change the library per request by passing in `LIB_MECHANIZE` or `LIB_REQUESTS` or you can change the library for the session by setting `browser.request_library` to either of those constants.

Parameters

- **url_or_object** – A full qualified URL or a Plone object (which has an `absolute_url` method). Defaults to the Plone Site URL.

- **data** (*dict or string*) – A dict with data which is posted using a *POST* request, or request payload as string.
- **view** (*string*) – The name of a view which will be added at the end of the current URL.
- **library** (LIB_MECHANIZE or LIB_REQUESTS) – Lets you explicitly choose the request library to be used for this request.
- **referer** (Boolean (Default False)) – Sets the referer when set to True.
- **method** (*string*) – The HTTP request method. Defaults to 'GET' when not set, unless data is provided, then its set to 'POST'.
- **headers** (*dict*) – A dict with custom headers for this request.
- **send_authenticator** (Boolean (Default False)) – When true, a plone.protect CSRF authenticator token is sent as if we would submit a prepared form. When this flag option is used, the request may be sent as POST. The code using the testbrowser with the send_authenticator option must make sure that plone.protect is installed.

See also:

visit()

See also:

LIB_MECHANIZE

See also:

LIB_REQUESTS

open_html(*html*)

Opens a HTML page in the browser without doing a request. The passed html may be a string or a file-like stream.

Parameters **html** (*string or file-like object*) – The HTML content to load in the browser.

Returns The browser object.

parse(*xml_or_html*)

Parse XML or HTML with the default parser. For XML mime types the XML parser is used, otherwise the HTML parser.

See also:

ftw.testbrowser.core.Browser.parse_as_html

See also:

ftw.testbrowser.core.Browser.parse_as_xml

Parameters **xml** (*string*) – The XML or HTML to parse.

parse_as_html(*html=None*)

Parse the response document with the HTML parser.

See also:

ftw.testbrowser.core.Browser.parse_as_xml

See also:

ftw.testbrowser.core.Browser.parse

Parameters **html** (*string*) – The HTML to parse (default: current response).

parse_as_xml(*xml=None*)
Parse the response document with the XML parser.

See also:

ftw.testbrowser.core.Browser.parse_as_html

See also:

ftw.testbrowser.core.Browser.parse

Parameters **xml** (*string*) – The XML to parse (default: current response).

reload()
Reloads the current page by redoing the previous requests with the same arguments. This applies for GET as well as POST requests.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns The browser object.

Return type *ftw.testbrowser.core.Browser*

replace_request_header(*name*, *value*)
Adds a permanent request header which is sent with every request. Before adding the request header all existing request headers with the same name are removed.

Parameters

- **name** (*string*) – Name of the request header
- **value** (*string*) – Value of the request header

See also:

replace_request_header()

See also:

clear_request_header()

reset()
Resets the browser: closes active sessions and resets the internal state.

root
The current document root node.

status_code
The status code of the last response or `None` when no request was done yet.

Type *int*

status_reason
The status reason of the last response or `None` when no request was done yet. Examples: `"OK"`, `"Not Found"`.

Type *string*

url
The URL of the current page.

visit (**args*, ***kwargs*)
Visit is an alias for *open()*.

See also:

open()

webdav (*method*, *url_or_object=None*, *data=None*, *view=None*, *headers=None*)
Makes a webdav request to the Zope server.

It is required that a `ZSERVER_FIXTURE` is used in the test setup (e.g. `PLONE_ZSERVER'' from ``plone.app.testing`).

Parameters

- **method** (*string*) – The HTTP request method (`OPTIONS`, `PROPFIND`, etc)
- **url_or_object** – A full qualified URL or a Plone object (which has an `absolute_url` method). Defaults to the Plone Site URL.
- **data** (*dict*) – A dict with data which is posted using a *POST* request.
- **view** (*string*) – The name of a view which will be added at the end of the current URL.
- **headers** (*dict*) – Pass in reqest headers.

xpath (***kwargs*)
Select one or more HTML nodes by using an *xpath* selector.

Parameters **xpath_selector** (*string*) – The xpath selector.

Returns Object containg matches.

Return type *ftw.testbrowser.nodes.Nodes*

4.1.1 Drivers

Drivers are responsible for making the request and responding to basic questions, such as the current URL or response headers.

RequestsDriver

class ftw.testbrowser.drivers.requestsdriver.**RequestsDriver** (*browser*)
Bases: `object`

The requests driver uses the "requests" library for making real requests.

append_request_header (*name*, *value*)
Add a new permanent request header which is sent with every request until it is cleared.

HTTP allows multiple request headers with the same name. Therefore this method does not replace existing names. Use `replace_request_header` for replacing headers.

Be aware that the `requests` library does not support multiple headers with the same name, therefore it is always a replace for the requests module.

Parameters

- **name** (*string*) – Name of the request header
- **value** (*string*) – Value of the request header

clear_request_header(*name*)
Removes a permanent header. If there is no such header, the removal is silently skipped.

Parameters **name** (*string*) – Name of the request header as positional argument

cloned(*subbrowser*)
When a browser was cloned, this method is called on each driver instance so that the instance can do whatever is needed to have a cloned state too.

Parameters **subbrowser** (*ftw.testbrowser.core.Browser*) – The cloned sub-browser instance.

get_response_body()
Returns the response body of the last response.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Response body

Return type string

get_response_cookies()
Retruns a dict-like object containing the cookies set by the last response.

Returns Response cookies dict

Return type dict

get_response_headers()
Returns a dict-like object containing the response headers. Returns an empty dict if there is no response.

Returns Response header dict

Return type dict

get_url()
Returns the current url, if we are on a page, or None.

Returns the current URL

Return type string or None

make_request(**args*, ***kwargs*)
Make a request to the url and return the response body as string.

Parameters

- **method** (*string*) – The HTTP request method, all uppercase.
- **url** (*string*) – A full qualified URL.
- **data** (*dict or string*) – A dict with data which is posted using a `POST` request, or the raw request body as a string.
- **headers** (*dict*) – A dict with custom headers for this request.
- **referer_url** – The referer URL or `None`.

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reload()
Reloads the current page by redoing the previous request with the same arguments. This applies for GET as well as POST requests.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reset()
Resets the driver: closes active sessions and resets the internal state.

MechanizeDriver

class ftw.testbrowser.drivers.mechdriver.**MechanizeDriver**(*browser*)
Bases: object

The mechanize driver uses the Mechanize browser with plone.testing integration for making the requests.

append_request_header(*name*, *value*)
Add a new permanent request header which is sent with every request until it is cleared.

HTTP allows multiple request headers with the same name. Therefore this method does not replace existing names. Use replace_request_header for replacing headers.

Be aware that the requests library does not support multiple headers with the same name, therefore it is always a replace for the requests module.

Parameters

- **name** (*string*) – Name of the request header
- **value** (*string*) – Value of the request header

clear_request_header(*name*)
Removes a permanent header. If there is no such header, the removal is silently skipped.

Parameters **name** (*string*) – Name of the request header as positional argument

cloned(*subbrowser*)
When a browser was cloned, this method is called on each driver instance so that the instance can do whatever is needed to have a cloned state too.

Parameters **subbrowser** (*ftw.testbrowser.core.Browser*) – The cloned subbrowser instance.

get_response_body()
Returns the response body of the last response.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Response body

Return type string

get_response_cookies()
Retruns a dict-like object containing the cookies set by the last response.

Returns Response cookies dict

Return type dict

get_response_headers()
Returns a dict-like object containing the response headers. Returns an empty dict if there is no response.

Returns Response header dict

Return type dict

get_url()
Returns the current url, if we are on a page, or None.

Returns the current URL

Return type string or None

make_request(**args*, ***kwargs*)
Make a request to the url and return the response body as string.

Parameters

- **method** (*string*) – The HTTP request method, all uppercase.
- **url** (*string*) – A full qualified URL.
- **data** (*dict or string*) – A dict with data which is posted using a `POST` request, or the raw request body as a string.
- **headers** (*dict*) – A dict with custom headers for this request.
- **referer_url** – The referer URL or `None`.

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reload()
Reloads the current page by redoing the previous request with the same arguments. This applies for GET as well as POST requests.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reset()
Resets the driver: closes active sessions and resets the internal state.

TraversalDriver

class ftw.testbrowser.drivers.traversaldriver.**TraversalDriver**(*browser*)
Bases: `object`

The traversal driver simulates requests by by calling the zope traversal directly. The purpose of the traversal driver is to be able to use a testbrowser in the same transaction / connection as the test code is run. This makes it possible to write browser tests without transactions.

append_request_header(*name*, *value*)
Add a new permanent request header which is sent with every request until it is cleared.

HTTP allows multiple request headers with the same name. Therefore this method does not replace existing names. Use `replace_request_header` for replacing headers.

Be aware that the `requests` library does not support multiple headers with the same name, therefore it is always a replace for the requests module.

Parameters

- **name** (*string*) – Name of the request header
- **value** (*string*) – Value of the request header

clear_request_header(*name*)
Removes a permanent header. If there is no such header, the removal is silently skipped.

Parameters **name** (*string*) – Name of the request header as positional argument

cloned(*subbrowser*)
When a browser was cloned, this method is called on each driver instance so that the instance can do whatever is needed to have a cloned state too.

Parameters **subbrowser** (*ftw.testbrowser.core.Browser*) – The cloned sub-browser instance.

get_response_body()
Returns the response body of the last response.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Response body

Return type string

get_response_cookies()
Retruns a dict-like object containing the cookies set by the last response.

Returns Response cookies dict

Return type dict

get_response_headers()
Returns a dict-like object containing the response headers. Returns an empty dict if there is no response.

Returns Response header dict

Return type dict

get_url()
Returns the current url, if we are on a page, or None.

Returns the current URL

Return type string or None

make_request(**args*, ***kwargs*)
Make a request to the url and return the response body as string.

Parameters

- **method** (*string*) – The HTTP request method, all uppercase.
- **url** (*string*) – A full qualified URL.
- **data** (*dict or string*) – A dict with data which is posted using a `POST` request, or the raw request body as a string.
- **headers** (*dict*) – A dict with custom headers for this request.
- **referer_url** – The referer URL or `None`.

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reload()
Reloads the current page by redoing the previous request with the same arguments. This applies for GET as well as POST requests.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reset()
Resets the driver: closes active sessions and resets the internal state.

StaticDriver

class ftw.testbrowser.drivers.staticdriver.**StaticDriver**(*browser*)
Bases: object

The static driver can load static HTML without doing an actual request. It does not support making requests at all.

append_request_header(*name*, *value*)
Add a new permanent request header which is sent with every request until it is cleared.

HTTP allows multiple request headers with the same name. Therefore this method does not replace existing names. Use replace_request_header for replacing headers.

Be aware that the requests library does not support multiple headers with the same name, therefore it is always a replace for the requests module.

Parameters

- **name** (*string*) – Name of the request header
- **value** (*string*) – Value of the request header

clear_request_header(*name*)
Removes a permanent header. If there is no such header, the removal is silently skipped.

Parameters **name** (*string*) – Name of the request header as positional argument

cloned(*subbrowser*)
When a browser was cloned, this method is called on each driver instance so that the instance can do whatever is needed to have a cloned state too.

Parameters **subbrowser** (*ftw.testbrowser.core.Browser*) – The cloned subbrowser instance.

get_response_body()
Returns the response body of the last response.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Response body

Return type string

get_response_cookies()
Retruns a dict-like object containing the cookies set by the last response.

Returns Response cookies dict

Return type dict

get_response_headers()
Returns a dict-like object containing the response headers. Returns an empty dict if there is no response.

Returns Response header dict

Return type dict

get_url()
Returns the current url, if we are on a page, or None.

Returns the current URL

Return type string or None

make_request(*method*, *url*, *data=None*, *headers=None*, *referer_url=None*)
Make a request to the url and return the response body as string.

Parameters

- **method** (*string*) – The HTTP request method, all uppercase.
- **url** (*string*) – A full qualified URL.
- **data** (*dict or string*) – A dict with data which is posted using a `POST` request, or the raw request body as a string.
- **headers** (*dict*) – A dict with custom headers for this request.
- **referer_url** – The referer URL or `None`.

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reload()
Reloads the current page by redoing the previous request with the same arguments. This applies for GET as well as POST requests.

Raises *ftw.testbrowser.exceptions.BlankPage*

Returns Status code, reason and body

Return type tuple: (int, string, string or stream)

reset()
Resets the driver: closes active sessions and resets the internal state.

4.2 Nodes and forms

4.2.1 Result set

class ftw.testbrowser.nodes.**Nodes**(**args*, ***kwargs*)
Bases: `list`

A list of HTML nodes. This is used as result set when doing queries (css / xpath) on the HTML document. It acts as a list.

css(***kwargs*)
Find nodes by a *css* expression which are within one of the nodes in this result set. The resulting nodes are merged into a new result set.

Parameters **css_selector** (*string*) – The CSS selector.

Returns Object containg matches.

Return type *ftw.testbrowser.nodes.Nodes*

find(**args*, ***kwargs*)
Find a elements by text. The elements are searched within each node of the current result set.

The method looks for: - a link with this text (normalized, including subelements' texts) - a field which has a label with this text - a button which has a label with this text

Parameters **text** (*string*) – The text to be looked for.

Returns A list of the parent of each node in the current result set.

Return type *ftw.testbrowser.nodes.Nodes*

first
The first element of the list.

Raises *ftw.testbrowser.exceptions.NoElementFound*

first_or_none
The first element of the list or `None` if the list is empty.

getparents()
Returns a list of each node's parent.

Returns The parent of each node of the current result set.

Return type *ftw.testbrowser.nodes.Nodes*

raw_text
A list of all `raw_text` properties of this result set.

See also:

ftw.testbrowser.nodes.NodeWrapper.raw_text()

Returns A list of raw text

Return type list of string

text
A list of all `text` properties of this result set.

See also:

ftw.testbrowser.nodes.NodeWrapper.text()

Returns A list of text

Return type list of string

text_content()
Returns a list with the text content of each node of this result set.

Returns A list of the *text_content* of each node.

Return type list

See also:

ftw.testbrowser.nodes.NodeWrapper.text_content()

xpath (***kwargs*)
Find nodes by an *xpath* expression which are within one of the nodes in this result set. The resulting nodes are merged into a new result set.

Parameters **xpath_selector** (*string*) – The xpath selector.

Returns Object containg matches.

Return type *ftw.testbrowser.nodes.Nodes*

4.2.2 Node wrappers

Node wrappers wrap the standard *lxml* elements and extend them with some useful methods so that it is nicely integrated in the *ftw.testbrowser* behavior.

class ftw.testbrowser.nodes.**NodeWrapper**(*node*, *browser*)
Bases: object

NodeWrapper is the default wrapper class in which each element will be wrapped for use in *ftw.testbrowser*. It wraps the elements returned by *lxml* and redirects calls if it does not overload them.

There are more specific node wrapper classes for some elements.

browser
The current browser instance.

classes
A list of css-classes of this element.

contains(*other*)
Test whether the passed *other* node is contained in the current node.

Parameters **other** (*ftw.testbrowser.nodes.NodeWrapper*) – The other node.

Returns *True* when *other* is within *self*.

Return type boolean

css(***kwargs*)
Find nodes within this node by a *css* selector.

Parameters **css_selector** (*string*) – The CSS selector.

Returns Object containg matches.

Return type *ftw.testbrowser.nodes.Nodes*

find(*text*)
Find an element by text within the current node.

The method looks for: - a link with this text (normalized, including subelements' texts) - a field which has a label with this text - a button which has a label with this text

Parameters

- **text** (*string*) – The text to be looked for.
- **within** (*ftw.testbrowser.nodes.NodeWrapper.*) – A node object for limiting the scope of the search.

Returns A single node object or *None* if nothing matches.

Return type *ftw.testbrowser.nodes.NodeWrapper*

innerHTML
The unmodified HTML content of the current node. The HTML-Tag of the current node is not included.

Returns HTML

Return type unicode

iterlinks(**args*, ***kwargs*)
Iterate over all links in this node. Each link is represented as a tuple with *node, attribute, link, pos*.

normalized_innerHTML
The whitespace-normalized HTML content of the current node. The HTML-Tag of the current node is not included. All series of whitespaces (including non-breaking spaces) are replaced with a single space.

Returns HTML

Return type unicode

normalized_outerHTML
The whitespace-normalized HTML of the current node and its children. The HTML-Tag of the current node is included. All series of whitespaces (including non-breaking spaces) are replaced with a single space.

Returns HTML

Return type unicode

outerHTML
The whitespace-normalized HTML of the current node and its children. The HTML-Tag of the current node is included.

Returns HTML

Return type unicode

parent(*css=None*, *xpath=None*)
Find the nearest parent which (optionally) does match a *css* or *xpath* selector.

If *parent* is called without an argument the first parent is returned.

Examples:

```
browser.css('.foo > .bar').first.parent('#content')
# equals
browser.css('.foo > .bar').first.parent(xpath='*[@id="content"]')
```

Parameters

- **css** (*string*) – The css selector.
- **xpath** (*string*) – The xpath selector.

Returns The parent node.

Return type *ftw.testbrowser.nodes.NodeWrapper*

raw_text
The original lxml raw text of this node.

Returns Original lxml raw text.

Return type unicode

text
Returns the whitespace-normalized text of the current node. This includes the text of each node within this node recursively. All whitespaces are reduced to a single space each, including newlines within the text.

HTML line breaks (<*br /*>) are turned into a single newlineand paragraphs (<*p*><*/p*>) and with two newlines, although the end of the string is stripped.

For having the original lxml raw text, use `raw_text`. .. seealso:: *ftw.testbrowser.nodes.NodeWrapper.raw_text()*

Returns The whitespace normalized text content.

Return type unicode

text_content()
Returns the text content of the current node, including the text content of each containing node recursively.

Returns The text content.

Return type unicode

within(*container*)
Test whether the passed *other* node contains the current node.

Parameters **other** (*ftw.testbrowser.nodes.NodeWrapper*) – The other node.

Returns *True* when *self* is within *other*.

Return type boolean

xpath(***kwargs*)
Find nodes within this node by a *css* selector.

Parameters **css_selector** (*string*) – The CSS selector.

Returns Object containg matches.

Return type *ftw.testbrowser.nodes.Nodes*

class ftw.testbrowser.nodes.**LinkNode**(*node*, *browser*)
Bases: *ftw.testbrowser.nodes.NodeWrapper*

Wrapps an <*a*> node.

click()
Clicks on the link, which opens the target in the current browser.

class ftw.testbrowser.nodes.**DefinitionListNode**(*node*, *browser*)
Bases: *ftw.testbrowser.nodes.NodeWrapper*

Wrapps a <*dl*> node.

definitions
Returns the normalized text of each <*dd*>-tag of this definition list.

Returns A list of text of each <*dd*>-node.

Return type list of unicode

items()
Returns a mapping (list with tuples) from <*dt*>-tags to <*dd*>-tags of this definition list.

Returns a dict where the key is the <*dt*>-node and the value is the <*dd*>-node.

Return type dict

items_text()
Returns a terms (<*dt*>) to definition (<*dd*>) mapping as list with tuples, each as normalized text.

Returns key is the text of the <*dt*>-node, value is the text of the <*dd*>-node.

Return type dict

keys()
Returns all <*dt*>-tags which are direct children of this definition list.

Returns A list of <*dt*>-tags.

Return type *ftw.testbrowser.nodes.Nodes*

terms
Returns the normalized text of each <*dt*>-tag of this definition list.

Returns A list of text of each <*dt*>-node.

Return type list of unicode

text_to_nodes()
Returns a dict with a mapping of text-terms to <*dd*>-nodes.

Returns key is the text of the <*dt*>-node, value is the <*dd*>-node.

Return type dict

values()
Returns all <*dd*>-tags which are direct children of this definition list.

Returns A list of <*dd*>-tags.

Return type *ftw.testbrowser.nodes.Nodes*

4.2.3 Forms, fields and widgets

class ftw.testbrowser.form.**Form**(*node*, *browser*)
Bases: *ftw.testbrowser.nodes.NodeWrapper*

action_url
The full qualified URL to send the form to. This should imitate normal browser behavior:

If the action is full qualified, use it as it is. If the action is relative, make it absolute by joining it with the page's base URL. If there is no action, the action is the current page URL.

The page's base URL can be set in the HTML document with a <base>-tag, otherwise the page URL is used.

field_label_to_name(*label*)
Accepts a field label (or a field name) and returns the field name of the field.

Parameters **label** (*string*) – The label of the field.

Returns The field name of the field.

Return type string

field_labels
A list of label texts and field names of each field in this form.

The list contains the whitespace normalized label text of each field. If there is no label or it has an empty text, the fieldname is used instead.

Returns A list of label texts (and field names).

Return type list of strings

field_labels_to_names(*values*)
Accepts a dict and converts its field labels (keys) to field names.

Parameters

- **values** (*dict*) – A dict of values where the keys are field labels.
- **values** – A dict of values where the keys are field names.

Return type dict

fill(*values*)
Accepts a dict, where the key is the name or the label of a field and the value is its new value and fills the form with theese values.

Parameters **values** (*dict*) – The key is the label or input-name and the value is the value to set.

Returns The form node.

Return type *ftw.testbrowser.form.Form*

find_button_by_label(**args*, ***kwargs*)
Finds a button of with a specific label in this form.

Parameters **label** (*string*) – The label of the button.

Returns The button node

Return type *ftw.testbrowser.nodes.NodeWrapper*

find_field(**args*, ***kwargs*)
Finds and returns a field by label or name.

Parameters **label_or_name** (*string*) – The label or the name of the field.

Returns The field node

Return type *ftw.testbrowser.nodes.NodeWrapper*

find_submit_buttons(**args*, ***kwargs*)
Returns all submit buttons of this form.

Returns a list of submit buttons

Return type *ftw.testbrowser.nodes.Nodes* of *ftw.testbrowser.form.SubmitButton* of

find_widget(*label*)
Finds a Plone widget (div.field) in a form.

Parameters **label** (*string*) – The label of the widget.

Returns Returns the field node or *None*.

Return type *ftw.testbrowser.nodes.NodeWrapper*

inputs
Returns a list of all input nodes of this form.

Returns All input nodes

Return type *ftw.testbrowser.nodes.Nodes*

save()
Clicks on the "Save" button in this form.

submit(*button=None*)
Submits this form by clicking on the first submit button. The behavior of click the first submit button is what browser usually do and may not get the expected results.

It might be more save to click the primary button specificall:

```
browser.find('Save').click()
# or
form.save()
```

See also:

save()

values
Returns the lxml *FieldsDict* of this form.

Returns lxml fields dict

Return type lxml.html.FieldsDict

class ftw.testbrowser.form.**TextAreaField**(*node*, *browser*)
Bases: *ftw.testbrowser.nodes.NodeWrapper*

The *TextAreaField* node wrapper wraps a text area field and makes sure that the TinyMCE widget finds its label, since the markup of the TinyMCE widget is not standard.

class ftw.testbrowser.form.**SubmitButton**(*node*, *browser*)
Bases: *ftw.testbrowser.nodes.NodeWrapper*

Wraps a submit button and makes it clickable.

click()
Click on this submit button, which makes the form submit with this button.

form
Returns the form of which this button is parent. It returns the first form node if it is a nested form.

Returns the form node

Return type *ftw.testbrowser.form.Form*

4.2.4 Tables

class ftw.testbrowser.table.**Table**(*node*, *browser*)
Bases: *ftw.testbrowser.nodes.NodeWrapper*

Represents a `table` tag.

body_rows
All body rows of this table. Body rows are those rows which are neither heading rows nor footer rows.

See also:

ftw.testbrowser.table.Table.head_rows(), *ftw.testbrowser.table.Table.foot_rows()*

Returns A list of body rows which are part of this table.

Return type *ftw.testbrowser.nodes.Nodes*

cells
All cells of this table.

Returns A list of cells which are part of this table.

Return type *ftw.testbrowser.nodes.Nodes*

column(*index_or_titles*, *head=True*, *body=True*, *foot=True*, *head_offset=0*, *as_text=True*)
Returns a list of values of a specific column. The column may be identified by its index (integer) or by the title (string).

Parameters

- **index_or_titles** (*int or string or list of strings*) – Index or title of column
- **head** (boolean (Default: `True`)) – Include head rows.
- **body** (boolean (Default: `True`)) – Include body rows.
- **foot** (boolean (Default: `True`)) – Include foot rows.
- **head_offset** (int (Default: `0`)) – Offset for the header for removing header rows.
- **as_text** (Boolean (Default: `True`)) – Converts cell values to text.

Returns A list of lists of texts.

Return type list

dicts(*body=True*, *foot=True*, *head_offset=0*, *as_text=True*)
Returns a list of dicts, where each dict is a row (of either table body or table foot). The keys of the row dicts are the table headings and the values are the cell texts. Cells with colspan are repeated.

Parameters

- **body** (boolean (Default: `True`)) – Include body rows.
- **foot** (boolean (Default: `True`)) – Include foot rows.
- **head_offset** (int (Default: `0`)) – Offset for the header for removing header rows.
- **as_text** (Boolean (Default: `True`)) – Converts cell values to text.

Returns A list of lists of texts.

Return type list

filter_unfamiliars(*nodes*)
Returns all nodes from the `nodes` list which are part of this table, filtering all other nodes (unfamiliars).

Parameters **nodes** – The list of nodes to filter.

Returns The filtered list of nodes.

Return type *ftw.testbrowser.nodes.Nodes*

find(*text*)
Find a cell of this table by text. When nothing is found, it falls back to the default `find` behavior.

See also:

ftw.testbrowser.nodes.NodeWrapper.find()

Parameters **text** (*string*) – The text to be looked for.

Returns A single node object or *None* if nothing matches.

Return type *ftw.testbrowser.nodes.NodeWrapper*

foot_rows
All footer rows of this table. Footer rows are those rows (`tr`) which are within the `tfoot` tag.

Returns A list of footer rows.

Return type *ftw.testbrowser.nodes.Nodes*

get_rows(*head=False*, *body=False*, *foot=False*, *head_offset=0*)
Returns merged head, body or foot rows. Set the keyword arguments to `True` for selecting the type of rows.

Parameters

- **head** (boolean (Default: `False`)) – Selects head rows.
- **body** (boolean (Default: `False`)) – Selects body rows.
- **foot** (boolean (Default: `False`)) – Selects foot rows.
- **head_offset** (int (Default: `0`)) – Offset for the header for removing header rows.

Returns A list of rows which are part of this table.

Return type *ftw.testbrowser.nodes.Nodes*

get_titles(*head_offset=0*)
Returns the titles (thead) of the table. If there are multiple table head rows, the cells of the rows are merged per column (with newline as separator).

Parameters **head_offset** (int (Default: `0`)) – Offset for the header for removing header rows.

Returns A list of table head texts per column.

Return type list

head_rows
All heading rows of this table. Heading rows are those rows (`tr`) which are within the `thead` tag.

Returns A list of heading rows.

Return type *ftw.testbrowser.nodes.Nodes*

is_familiar(*node*)
Returns `True` when `node` is a component of the this table. Returns `False` when `node` is a table component but is part of a nested table.

Parameters **node** (*ftw.testbrowser.nodes.NodeWrapper*) – The node to check.

Returns whether `node` is part of this table

Return type boolean

lists(*head=True*, *body=True*, *foot=True*, *head_offset=0*, *as_text=True*)
Returns a list of lists, where each list represents a row and contains the texts of the cells. Cells with colspan are repeated (padding) so that row lengths are equal.

Parameters

- **head** (boolean (Default: `True`)) – Include head rows.
- **body** (boolean (Default: `True`)) – Include body rows.
- **foot** (boolean (Default: `True`)) – Include foot rows.
- **head_offset** (int (Default: `0`)) – Offset for the header for removing header rows.
- **as_text** (Boolean (Default: `True`)) – Converts cell values to text.

Returns A list of lists of texts.

Return type list

rows
All rows of this table.

Returns A list of rows which are part of this table.

Return type *ftw.testbrowser.nodes.Nodes*

titles
Returns the titles (thead) of the table. If there are multiple table head rows, the cells of the rows are merged per column (with newline as separator).

Returns A list of table head texts per column.

Return type list

class ftw.testbrowser.table.**TableCell**(*node*, *browser*)
Bases: *ftw.testbrowser.table.TableComponent*

Represents a table cell (`td` or `th`).

row
Returns the row (`tr` node) of this cell.

Returns The row node.

Return type *ftw.testbrowser.table.TableRow*

class ftw.testbrowser.table.**TableComponent**(*node*, *browser*)
Bases: *ftw.testbrowser.nodes.NodeWrapper*

Represents any component of a table tag. This includes: 'colgroup', 'col', 'thead', 'tbody', 'tfoot', 'tr', 'td', 'th'

table
Returns the table of which this button is parent. It returns the first table node if it is a nested table.

Returns the table node

Return type *ftw.testbrowser.table.Table*

class ftw.testbrowser.table.**TableRow**(*node*, *browser*)
Bases: *ftw.testbrowser.table.TableComponent*

Represents a table row (`tr`).

cells
The cell nodes of this row.

Returns A `Node` list of cell nodes.

Return type *ftw.testbrowser.nodes.Nodes*

dict()
Returns this row as dict. The keys of the dict are the column titles of the table, the values are the cell texts of this row.

Returns A dict with the cell texts.

Return type dict

ftw.testbrowser.table.**colspan_padded_text**(*row*)
Returns a list with the normalized_text of each cell of the `row`, but adds empty padding-cells for cells with a colspan.

Parameters **node** (`ftw.testbrowser.nodes.TableRow`) – The row node.

Returns A list of cell texts

Return type list

4.3 Page objects

4.3.1 Plone page object

`ftw.testbrowser.pages.plone.`**`document_description`**(*browser=<ftw.browser.core.Browser instance>*)

Returns the whitespace-normalized document description of the current page or None.

`ftw.testbrowser.pages.plone.`**`first_heading`**(*browser=<ftw.browser.core.Browser instance>*)

Returns the whitespace-normalized first heading of the current page.

`ftw.testbrowser.pages.plone.`**`logged_in`**(*browser=<ftw.browser.core.Browser instance>*)

If a user is logged in in the current browser session (last request), it resturns the user-ID, otherwise `False`.

`ftw.testbrowser.pages.plone.`**`portal_type`**(*browser=<ftw.browser.core.Browser instance>*)

Returns the current content type, extracted from the body css classes.

`ftw.testbrowser.pages.plone.`**`view`**(*browser=<ftw.browser.core.Browser instance>*)

Returns the view, taken from the template class, of the current page.

`ftw.testbrowser.pages.plone.`**`view_and_portal_type`**(*browser=<ftw.browser.core.Browser instance>*)

Returns a tuple of the view and the content type, both taken from the body css classes.

4.3.2 Editbar page object

4.3.3 Factories menu page object

`ftw.testbrowser.pages.factoriesmenu.`**`add`**(*type_name*, *browser=<ftw.browser.core.Browser instance>*)

Clicks on the add-link in the factories menu for the passed type name. The type name is the literal link label. This opens the add form for this type.

Parameters **type_name** (*string*) – The name (label) of the type to add.

ftw.testbrowser.pages.factoriesmenu.**addable_types**(*browser=<ftw.browser.core.Browser instance>*)
Returns a list of addable types. Each addable types is the link label in the factories menu.

ftw.testbrowser.pages.factoriesmenu.**menu**(*browser=<ftw.browser.core.Browser instance>*)
Returns the factories menu container node or `None` if it is not visible.

ftw.testbrowser.pages.factoriesmenu.**visible**(*browser=<ftw.browser.core.Browser instance>*)
Returns `True` when the factories menu is visible on the current page.

4.3.4 Status messages page object

ftw.testbrowser.pages.statusmessages.**as_string**(*filter_=None*, *browser=<ftw.browser.core.Browser instance>*)
All status messages as string instead of dict, so that it can be used for formatting assertion errors. Pass a type ("info", "warning" or "error") for filter_ing the messages.

ftw.testbrowser.pages.statusmessages.**assert_message**(*text*, *browser=<ftw.browser.core.Browser instance>*)
Assert that a status message is visible.

ftw.testbrowser.pages.statusmessages.**assert_no_error_messages**(*browser=<ftw.browser.core.Browser instance>*)
Assert that there are no error messages.

ftw.testbrowser.pages.statusmessages.**assert_no_messages**(*browser=<ftw.browser.core.Browser instance>*)
Assert that there are no status messages at all.

ftw.testbrowser.pages.statusmessages.**error_messages**(*browser=<ftw.browser.core.Browser instance>*)
Returns all "error" statusmessages.

ftw.testbrowser.pages.statusmessages.**info_messages**(*browser=<ftw.browser.core.Browser instance>*)
Returns all "info" statusmessages.

ftw.testbrowser.pages.statusmessages.**messages**(*browser=<ftw.browser.core.Browser instance>*)
Returns a dict with lists of status messages (normalized text) for "info", "warning" and "error".

ftw.testbrowser.pages.statusmessages.**warning_messages**(*browser=<ftw.browser.core.Browser instance>*)
Returns all "warning" statusmessages.

4.3.5 dexterity page object

ftw.testbrowser.pages.dexterity.**erroneous_fields**(*browser=<ftw.browser.core.Browser instance>*)
Returns a mapping of erroneous fields (key is label or name of the field) to a list of error messages for the fields on a dexterity add and edit forms (*form#form*).

Returns A dict of erroneous fields with error messages.

Return type dict

4.3.6 z3cform page object

ftw.testbrowser.pages.z3cform.**erroneous_fields**(*form*, *browser=<ftw.browser.core.Browser instance>*)

Returns a mapping of erroneous fields (key is label or name of the field) to a list of error messages for the fields on the form passed as argument.

Parameters

- **form** (*ftw.testbrowser.form.Form*) – The form node to check for errors.
- **browser** (*ftw.testbrowser.core.Browser*) – A browser instance. (Default: global browser)

Returns A dict of erroneous fields with error messages.

Return type dict

4.3.7 Folder contents page object

4.4 Exceptions

exception ftw.testbrowser.exceptions.**AmbiguousFormFields**

Bases: *ftw.testbrowser.exceptions.BrowserException*

Trying to change fields over multiple forms is not possible.

exception ftw.testbrowser.exceptions.**BlankPage**(*message=''*)

Bases: *ftw.testbrowser.exceptions.BrowserException*

The browser is on a blank page.

exception ftw.testbrowser.exceptions.**BrowserException**

Bases: exceptions.Exception

ftw.testbrowser exception base class.

exception ftw.testbrowser.exceptions.**BrowserNotSetUpException**

Bases: *ftw.testbrowser.exceptions.BrowserException*

The browser is not set up properly. Use the browser as a context manager with the "with" statement.

exception ftw.testbrowser.exceptions.**ContextNotFound**(*message=None*)

Bases: *ftw.testbrowser.exceptions.BrowserException*

When trying to access a context but the current page has no context information, this exception is raised.

exception ftw.testbrowser.exceptions.**FormFieldNotFound**(*label_or_name*, *labels=None*)

Bases: *ftw.testbrowser.exceptions.BrowserException*

Could not find a form field.

exception ftw.testbrowser.exceptions.**HTTPClientError**(*status_code*, *status_reason*)

Bases: *ftw.testbrowser.exceptions.HTTPError*

The request caused a client error with status codes 400-499.

Variables

- ***status_code*** – The status code number, e.g. `404`
- ***status_reason*** – The status reason, e.g. `"Not Found"`

exception ftw.testbrowser.exceptions.**HTTPError**(*status_code*, *status_reason*)
Bases: exceptions.IOError

The request has failed.

Variables

- ***status_code*** – The status code number
- ***status_reason*** – The status reason.

exception ftw.testbrowser.exceptions.**HTTPServerError**(*status_code*, *status_reason*)
Bases: *ftw.testbrowser.exceptions.HTTPError*

The request caused a server error with status codes 500-599.

Variables

- ***status_code*** – The status code number, e.g. `500`
- ***status_reason*** – The status reason, e.g. `"Internal Server Error"`

exception ftw.testbrowser.exceptions.**InsufficientPrivileges**(*status_code*, *status_reason*)
Bases: *ftw.testbrowser.exceptions.HTTPClientError*

This exception is raised when Plone responds that the user has insufficient privileges for performing that request.

Plone redirects the user to a "require_login" script when the user has not enough privileges (causing an internal Unauthorized exception to be raised).

When a user is logged in, the "require_login" script will render a page with the title "Insufficient Privileges". For anonymous users, the login form is rendered. Both cases cause the testbrowser to raise an InsufficientPrivileges exception.

This exception can be disabled with by disabling the `raise_http_errors` option.

exception ftw.testbrowser.exceptions.**NoElementFound**(*query_info=None*)
Bases: *ftw.testbrowser.exceptions.BrowserException*

Empty result set has no elements.

exception ftw.testbrowser.exceptions.**NoWebDAVSupport**
Bases: *ftw.testbrowser.exceptions.BrowserException*

The current testbrowser driver does not support webdav requests.

exception ftw.testbrowser.exceptions.**OnlyOneValueAllowed**
Bases: *ftw.testbrowser.exceptions.BrowserException*

The field or widget does not allow to set multiple values.

exception ftw.testbrowser.exceptions.**OptionsNotFound**(*field_label*, *options*, *labels=None*)
Bases: *ftw.testbrowser.exceptions.BrowserException*

Could not find the options for a widget.

exception ftw.testbrowser.exceptions.**RedirectLoopException**(*url*)
Bases: *ftw.testbrowser.exceptions.BrowserException*

The server returned a redirect response that would lead to an infinite redirect loop.

exception ftw.testbrowser.exceptions.**ZServerRequired**
Bases: *ftw.testbrowser.exceptions.BrowserException*

The requests driver can only be used with a running ZServer. Use the *plone.app.testing.PLONE_ZSERVER* testing layer.

CHAPTER 5

Changelog

5.1 2.1.2 (2020-07-28)

- editbar.menu and editbar.menus can now find workflow contentmenu in Plone5. [djowett-ftw]

5.2 2.1.1 (2020-02-19)

- Since the default mode changed in python3 [Portingguide] we should use `from io import open` to make it compatible with both versions in `core.py`. This is relevant for e.g. `browser.debug()` [busykoala]

5.3 2.1.0 (2019-12-04)

- Implement webtest driver. [buchi]

5.4 2.0.0 (2019-12-04)

- Add support for Plone 5.2. [buchi]
- Make Traversal and Mechanize drivers optional as the underlying libraries are no longer available in Zope 4 (Plone 5.2). [buchi]
- No longer use plone.formwidget.autocomplete and plone.formwidget.contenttree in tests with Plone 5 and later. [buchi]
- Implement AjaxSelectWidget and improve RelatedItemsWidget. [buchi]
- Modernize z3c form used in test by using directives to setup widgets and removing form wrapper. [buchi]
- Plone 5: disable resource registries in order to improve overall performance. [jone]

- Add support for Python 3. [buchi]

5.5 1.30.1 (2019-01-25)

- Add an *allow_redirects* property to *browser* for being able to block following redirects. [Rotonen]

5.6 1.30.0 (2018-08-03)

- Fix encoding problems for HTML documents without encoding declaration by respecting the encoding in the content-type response header. [jone]

5.7 1.29.7 (2018-04-30)

- Fix namespacing issues in WebDav Documents generated by Plone. [njohner]

5.8 1.29.6 (2018-03-26)

- Fix MIME encoding error when uploading files and using unicode form values. [jone]

5.9 1.29.5 (2017-12-04)

- Improve extraction of erroneous fields. [jone, mbaechtold]

5.10 1.29.4 (2017-11-30)

- Fix unbound local error. [deiferni]

5.11 1.29.3 (2017-11-23)

- Add missing dependency on plone.uuid. [njohner]

5.12 1.29.2 (2017-11-10)

- Fix browser.debug to finish writing file before opening. [jone]

5.13 1.29.1 (2017-11-08)

- Improve opening relative URLs. [mbaechtold]

5.14 1.29.0 (2017-10-13)

- Refactor plone.protect auto CSRF support for compatibility with ftw.testing's COMPONENT_REGISTRY_ISOLATION isolation layer. [jone]

5.15 1.28.1 (2017-10-03)

- Factories menu: fix regression when menu visible but empty. [jone]

5.16 1.28.0 (2017-10-03)

- Let traversal driver support plone.protect>=3 auto CSRF protection. [jone]
- Add Plone 5.1 support. [jone, maethu]

5.17 1.27.0 (2017-09-15)

- Improve multi browser support in standard page objects. [jone]

5.18 1.26.3 (2017-09-11)

- Make `plone.protect` import optional so that `ftw.testbrowser` works without `plone.protect`. [jone]

5.19 1.26.2 (2017-08-14)

- Disallow using browser as nested context manager. [jone]
- Fix reset behavior while used in context manager. [jone]

5.20 1.26.1 (2017-07-31)

- Datagridfield widget: support cell-filling by other widgets. [jone]

5.21 1.26.0 (2017-07-27)

- Feature: raise `InsufficientPrivileges` when a Plone request causes a insufficient privileges problem. [jone]

5.22 1.25.0 (2017-07-04)

- Add editbar page object module. [jone]
- Reimplement internal query info with a QueryInfo class. [jone]
- Add send_authenticator option for CSRF support. [jone]

5.23 1.24.3 (2017-06-23)

- Fix issues with read the docs builds. [jone]

5.24 1.24.2 (2017-06-21)

- Traversal driver: decode gzip encoded responses. [jone]

5.25 1.24.1 (2017-06-19)

- Declare missing dependencies. [lgraf]

5.26 1.24.0 (2017-06-16)

- Log exceptions to stderr when they are not expected. [jone]
- Standardize redirect loop detection: always throw a `RedirectLoopException`. [jone]
- Add traversal request driver. [jone]

5.27 1.23.2 (2017-06-16)

- Fix *browser.context* when base_url ends with a view name. [phgross]

5.28 1.23.1 (2017-05-02)

- Fix *browser.debug* when body is a bytestring. [jone]

5.29 1.23.0 (2017-04-28)

- Introduce `browser.expect_unauthorized` context manager. [jone]

5.30 1.22.2 (2017-04-28)

- HTTPError: include code and reason in exception. [jone]
- Docs: Fix wrong expect_http_error argument names. [jone]

5.31 1.22.1 (2017-04-28)

- Docs: swith to RTD, update URLs. [jone]
- Docs: Switch to RTD Sphinx theme. [lgraf]

5.32 1.22.0 (2017-04-28)

- Forbid setting of "x-zope-handle-errors" header. [jone]
- Add an option `browser.exception_bubbling`, disabled by default. [jone]
- Mechanize: no longer disable "x-zope-handle-errors". [jone]
- Introduce `browser.expect_http_error()` context manager. [jone]
- Add an option `browser.raise_http_errors`, enabled by default. [jone]
- Raise `HTTPClientError` and `HTTPServerError` by default. [jone]
- Introduce `browser.status_reason`. [jone]
- Introduce `browser.status_code`. [jone]

5.33 1.21.0 (2017-04-19)

- Make `zope.globalrequest` support optional. [jone]
- Add testing layers for setting the default driver. [jone]
- Add `default_driver` option to the driver. [jone]
- Refactoring: introduce request drivers. [jone]

5.34 1.20.0 (2017-04-10)

- Add Support for Button tag. [tschanzt]
- No longer test with Archetypes, test only with dexterity. [jone]
- Support latest Plone 4.3.x release. [mathias.leimgruber]

5.35 1.19.3 (2016-07-25)

- Declare some previously missing test requirements. [lgraf]
- Declare previously missing dependency on zope.globalrequest (introduced in #35). [lgraf]

5.36 1.19.2 (2016-06-27)

- Preserve the request of zope.globalrequest when opening pages with mechanize. [deiferni]
- Also provide advice for available options in exception message. [lgraf]

5.37 1.19.1 (2015-08-20)

- Preserve radio-button input when filling forms with radio buttons. [deiferni]

5.38 1.19.0 (2015-07-31)

- Implement browser.click_on(tex) short cut for clicking links. [jone]
- Fix encoding error in assertion message when selecting a missing select option. [mbaechtold]

5.39 1.18.1 (2015-07-23)

- Fix GET form submission to actually submit it with GET. [jone]

5.40 1.18.0 (2015-07-22)

- Table: add new ".column" method for getting all cells of a column. [jone]

5.41 1.17.0 (2015-07-22)

- Add support for filling `collective.z3cform.datagridfield`. [jone, mbaechtold]

5.42 1.16.1 (2015-07-13)

- Autocomplete widget: extract URL from javascript. [jone]

5.43 1.16.0 (2015-07-08)

- Add image upload widget support (archetypes and dexterity). [jone]

5.44 1.15.0 (2015-05-07)

- Parse XML responses with XML parser instead of HTML parser. New methods for parsing the response: `parse_as_html`, `parse_as_xml` and `parse`. [jone]
- Add browser properties `contenttype`, `mimetype` and `encoding`. [jone]

5.45 1.14.6 (2015-04-17)

- Use `cssselect` in favor of `lxml.cssselect`. This allows us to use `lxml >= 3`. [jone]
- Added tests for z3c date fields. [phgross]

5.46 1.14.5 (2015-01-30)

- AutocompleteWidget: Drop query string from base URL when building query URL. [lgraf]

5.47 1.14.4 (2014-10-03)

- Widgets: test for sequence widget after testing for autocomplete widgets. Some widgets match both, autocomplete and sequence widgets. In this case we want to have the autocomplete widget. [jone]

5.48 1.14.3 (2014-10-02)

- Fix error with textarea tags without id-attributes. [jone]

5.49 1.14.2 (2014-09-29)

- Fix an issue with relative urls. [jone, deiferni]

5.50 1.14.1 (2014-09-26)

- Set the HTTP `REFERER` header correctly. [jone]

5.51 1.14.0 (2014-09-26)

- Add folder_contents page object. [jone]
- Update table methods with keyword arguments:
  - head_offset: used for stripping rows from the header
  - as_text: set to False for getting cell nodes

  [jone]

5.52 1.13.4 (2014-09-22)

- Filling selects: verbose error message when option not found. The available options are now included in the message. [jone]

5.53 1.13.3 (2014-09-02)

- Node.text: remove multiple spaces in a row caused by nesting. [jone]

5.54 1.13.2 (2014-08-06)

- Fix problems when filling forms which have checked checkedbox. [phgross]

5.55 1.13.1 (2014-07-15)

- Fix encoding problem on binary file uploads. [jone]

5.56 1.13.0 (2014-06-12)

- Add a Dexterity namedfile upload widget. [lgraf]

5.57 1.12.4 (2014-05-30)

- Fix python 2.6 support. [jone]

5.58 1.12.3 (2014-05-30)

- Fix z3cform choice collection widget to support Plone < 4.3. [jone]

5.59 1.12.2 (2014-05-29)

- Fix z3cform choice collection widget submit value. The widget creates hidden input fields on submit. [jone]

5.60 1.12.1 (2014-05-29)

- Fix error in z3cform choice collection widget when using paths. [jone]

5.61 1.12.0 (2014-05-29)

- Add a z3cform choice collection widget. This is used for z3cform List fields with Choice value_type. [jone]
- Add select field node wrapper with methods for getting available options. [jone]

5.62 1.11.4 (2014-05-22)

- browser.open(data): support multiple values for the same data name. The values can either be passed as a dict with lists as values or as a sequence of two-element tuples. [jone]

5.63 1.11.3 (2014-05-19)

- Fix browser.url regression when the previous request raised an exception. [jone]

5.64 1.11.2 (2014-05-17)

- Make NoElementFound exception message more verbose. When a *.first* on an empty result set raises a NoElementFound exception, the exception message now includes the original query. [jone]

5.65 1.11.1 (2014-05-17)

- Fix browser cloning regression in autocomplete widget "query". The cloned browser did no longer have the same headers / cookies, causing authenticated access to be no longer possible. [jone]
- New browser.clone method for creating browser clones. [jone]
- Update standard page objects to accept browser instace as keyword arguments. This makes it possible to use the page objects with non-standard browsers. [jone]

5.66 1.11.0 (2014-05-14)

- New browser.base_url property, respecting the <base> tag. [jone]
- New browser.debug method, opening the current page in your real browser. [jone]
- New browser.on method, a lazy variant of browser.open. [jone]
- New browser.reload method, reloading the current page. [jone]
- Improve requests library support:
  - Support choosing requests library, make Zope app setup optional. When no Zope app is set up, the `requests` library is set as default, otherwise `mechanize`.
  - Support form submitting with requests library.
  - Improve login and header support for equests library requests.
  - Add browser.cookies support for requests library requests.
  - Use requests library sessions, so that cookies and headers persist.
  - Automatically use "POST" when data is submitted.

  [jone]
- Login improvements:

  - Support passing member objects to browser.login(). The users / members are still expected to hav TEST_USER_PASSWORD as password.
  - Refactor login to use the new request header methods.

  [jone]

- Add request header methods for managing permanent request headers:
  - browser.append_request_header
  - browser.replace_request_header
  - browser.clear_request_header

  [jone]

- Refactor Form: eliminate class methods and do not use the global browser. This improves form support when running multiple browser instances concurrently.
  - Form.field_labels (class method) is now a instance property and public API.
  - Form.find_widget_in_form (class method) is removed and replaced with Form.find_widget (instance method).
  - Form.find_field_in_form (class method) is removed and replaced Form.get_field (instance method).
  - Form.find_form_element_by_label_or_name (class method) is removed and replaced with browser.find_form_by_field.
  - Form.find_form_by_labels_or_names (class method) is removed and replaced with browser.find_form_by_fields.
  - New Form.action_url property with the full qualified action URL.
  - Fix form action URL bug when using relative paths in combination with document-style base url.

  [jone]

- Fix wrapping input.label - this did only work for a part of field types. [jone]
- Fix UnicodeDecodeError in node string representation. [mathias.leimgruber]

5.67 1.10.0 (2014-03-19)

- Add NodeWrapper-properties:
  - innerHTML
  - normalized_innerHTML
  - outerHTML
  - normalized_outerHTML

  [jone, elioschmutz]

5.68 1.9.0 (2014-03-18)

- Add support for filling AT MultiSelectionWidget. [jone]

5.69 1.8.0 (2014-03-04)

- Add a `context` property to the browser with the current context (Plone object) of the currently viewed page. [jone]

5.70 1.7.3 (2014-02-28)

- Fix encoding problem in factories menu page object. The problem occured when having a "Restrictions..." entry in the menu. [jone]

5.71 1.7.2 (2014-02-25)

- Form: Support checking checkboxes without a value. Checkboxes without a value attribute are invalid but common. The default browser behavior is to fallback to the value "on". [jone]

5.72 1.7.0 (2014-02-03)

- ContentTreeWidget: support filling objects as values. [jone]

5.73 1.6.1 (2014-01-31)

- Implement *logout* on browser, logout before each login. [jone]

5.74 1.6.0 (2014-01-29)

- Add *cookies* property to the browser. [jone]

5.75 1.5.3 (2014-01-28)

- Fix multiple wrapping on browser.forms. [jone]

5.76 1.5.2 (2014-01-17)

- Implement archetypes datetime widget form filling. [jone]

5.77 1.5.1 (2014-01-07)

- Fix encoding problems when posting unicode data directly with Browser.open. [jone]
- Support form filling with bytestrings. [jone]

- Fix form filling with umlauts. [jone]
- Fix form fill for single select fields. [jone]

5.78 1.5.0 (2014-01-03)

- Implement AT file upload widget, because the label does not work. [jone]
- Implement file uploads. [jone]
- Add "headers" property on the browser. [jone]

5.79 1.4.0 (2013-12-27)

- Deprecate *normalized_text* method, replace it with *text* property. The *text* property is more intuitive and easier to remember. The *text* property has almost the same result as *normalized_text*, but it represents <*br/*> and <*p*> with single and double newlines respectively. *text* is to be the lxml *text* property, which contained the raw, non-recursive text of the current node and is now available as *raw_text* property. [jone]
- open_html: make debugging file contain passed HTML. [jone]
- Sequence widget: implement custom form filling with label support and validation. [jone]
- Sequence widget: add additional properties with inputs and options. [jone]

5.80 1.3.0 (2013-12-11)

- Implement "query" method on autocomplete widget. [jone]
- Implement form fill for z3cform datetime widget. [jone]
- Fix setting attributes on nodes when wrapped with NodeWrapper. [jone]
- Implement form fill for z3cform autocomplete widgets. [jone]
- Implement form fill for z3cform sequence widgets. [jone]
- Add `webdav` method for doing WebDAV requests with a ZServer. [jone]

5.81 1.2.0 (2013-11-24)

- Add *open_html* method to browser object, allowing to pass in HTML directly. [jone]

5.82 1.1.0 (2013-11-07)

- Add dexterity page object, refactor z3cform page object. [jone]
- Add table nodes with helpers for table testing. [jone]
- Merging "Nodes" lists returns a new "Nodes" list, not a "list". [jone]
- Show containing elements in string representation of "Nodes" list. [jone]

- Fix direct child selection with CSS (node.css(">tag")). [jone]
- Add a `recursive` option to `normalized_text`. [jone]

5.83 1.0.2 (2013-10-31)

- When normalizing whitespaces, do also replace non-breaking spaces. [jone]

5.84 1.0.1 (2013-10-31)

- Add `first_or_none` property to `Nodes`. [jone]

5.85 1.0.0 (2013-10-28)

- Initial implementation. [jone]

CHAPTER 6

Links

- Source code on github: https://github.com/4teamwork/ftw.testbrowser
- Releases on pypi: https://pypi.python.org/pypi/ftw.testbrowser
- Issues on github: https://github.com/4teamwork/ftw.testbrowser/issues
- Continuous integration: https://jenkins.4teamwork.ch/search?q=ftw.testbrowser

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z