wheezy.html documentation

Release latest

Andriy Kornatskyy

Apr 17, 2021

Contents

CHAPTER 1

Introduction

wheezy.html is a python package written in pure Python code. It is a lightweight html widget library. Integrates with the following template systems:

- Jinja2 Templates
- Mako Templates
- Tenjin Templates
- Wheezy Templates

It is optimized for performance, well tested and documented.

Resources:

- source code and issues tracker are available on github
- documentation

CHAPTER 2

Contents

2.1 Getting Started

2.1.1 Install

wheezy.html requires python version 2.4 to 2.7 or 3.2+. It is operating system independent. You can install it from the pypi site:

```
$ pip install wheezy.html
```

2.2 Examples

Before we proceed let's setup a virtualenv environment, activate it and install:

```
$ pip install wheezy.html
```

2.2.1 Signin Widget

Suppose we need to add user credential input to an HTML form. In case of error it would be good to display it next to each input. Domain model looks like this:

```
class Credential(object):

    def __init__(self):
        self.username = ''
        self.password = ''
```

Here is what we can get in html template:

```
>>> from wheezy.html import widget
>>> credential = Credential()
>>> errors = {}
>>> credential = widget(credential, errors)
>>> credential.username.label('Username:')
<label for="username">Username:</label>
>>> credential.username.textbox(autocomplete='off')
<input autocomplete="off" type="text" id="username"
  value="" name="username" />
>>> credential.username.error()
```

Look how this changes in case of errors:

```
>>> errors = {'username': ['Required field cannot be left blank.']}
>>> credential = widget(credential, errors)
>>> credential.username.label('Username:')
<label class="error" for="username">Username:</label>
>>> credential.username.textbox(autocomplete='off')
<input name="username" value="" autocomplete="off"
   class="error" type="text" id="username" />
>>> credential.username.error()
<span class="error">Required field cannot be left blank.</span>
```

General error message:

```
>>> errors = {'__ERROR__': ['The username or password provided is incorrect.']}
>>> credential = widget(credential, errors)
>>> credential.error()
<span class="error-message">The username or password
  provided is incorrect.</span>
```

2.3 User Guide

2.3.1 Widget Name

Each attribute in widget corresponds to an appropriate attribute in `model`. Attribute name becomes `name` in the html element. By convention in html `id` underscore is replaced with dash. So attribute `confirm_password` remains unchanged in html name, however id will be `confirm-password`.

2.3.2 Widget Rendering

Once we know the name of the html widget, next we pass control to the appropriate widget for rendering:

```
credential.username.textbox(autocomplete='off')
```

Let's explain this single line:

- `credential` - domain object.
- `username` - attribute name of our domain object.
- `textbox` - widget we need to render.
- `autocomplete` - html specific attribute.

Once that code is executed we get the following:

```
<input autocomplete="off" type="text" id="username"
  value="" name="username" />
```

2.3.3 Value Formatting

You can the format model value before it is passed to widget for rendering.

Let's declare our domain model:

```
from datetime import date

class Registration(object):

    def __init__(self):
        self.date_of_birth = date.min
```

Here is how you can apply formatting:

```
registration.date_of_birth.format('%Y/%m/%d')
```

or this way:

```
registration.date_of_birth.format(
        format_provider=lambda value, ignore: value.strftime('%m-%d-%y'))
```

Widget formatting can followed by the actual widget that needs to be rendered:

```
registration.date_of_birth.format('%Y/%m/%d').textbox()
```

`format_provider` - a callable of the following form:

```
def my_format_provider(value, format_string):
    return value_formatted
```

There are default format providers for built-in types. You can replace and extend them with your own, by altering `format_providers` map:

```
from wheezy.html.utils import format_providers

format_providers['my_type'] = my_format_provider
```

Default implementation for date/time types formats its minimal value to an empty string.

2.3.4 Model Error

Since `widget` is initialized with model and errors, it is capable of decorating html widgets with attributes specific to errors. Let's see this in the following example:

```
errors = {'username': ['Required field cannot be left blank.']}
```

We get the errors from some sort of validation. The same `textbox` is now decorated with class error:

```
<input name="username" value="" autocomplete="off"
   class="error" type="text" id="username" />
```

So I can apply appropriate css style to draw a border around input, or what ever else, since in html I have distinguished between input with error and valid input.

Now let display error:

```
credential.username.error()
```

Read above as render error message for username, here is what we get:

```
<span class="error">Required field cannot be left blank.</span>
```

2.3.5 General Error

General error is not related to certain model attribute but is operation related instead. If `errors` dictionary contains an element with key __ERROR__ than that one is used as a general error:

```
errors = {'__ERROR__': 'The username or password provided is incorrect.'}
```

You can display it this way:

```
credential.error()
```

It renders the following html element only if the __ERROR__ key exists:

```
<span class="error-message">The username or password
  provided is incorrect.</span>
```

Notice class `error-message`. Your application is able to distinguish field errors from general errors.

2.3.6 Integration

wheezy.html integrates with the following template systems:

- Jinja2 Templates
- Mako Templates
- Tenjin Templates
- Wheezy Templates

Jinja2

wheezy.html integration with `Jinja2` is provided via the extension feature. Here is how to add *`WidgetExtension()`* to your code:

```
from wheezy.html.ext.jinja2 import WidgetExtension

env = Environment(
        ...
        extensions=[WidgetExtension])
```

The only thing *WidgetExtension()* does is translation of widget code to adequate Jinja2 code.

Let's demonstrate this with an example:

```
{{ model.remember_me.checkbox() }}
```

is translated to the following Jinja2 code (during template compilation phase):

```
<input id="remember-me" name="remember_me" type="checkbox"
value="1"
{% if 'remember_me' in errors: %}
 class="error"
{% endif %}
{% if  model.remember_me: %}
 checked="checked"
{% endif %} />
```

which effectively renders the HTML at runtime:

```
<input id="remember-me" name="remember_me" type="checkbox" value="1" />
```

Since widgets also decorate appropriate HTML tags in case of error, the errors dictionary must be available in the Jinja2 context:

```
template = env.get_template(template_name)
assert 'errors' in kwargs
template.render(
            **kwargs
)
```

See *wheezy.html.ext.jinja2* for more examples.

Mako

wheezy.html integration with Mako is provided via the preprocessor feature. Here is how to add widget_preprocessor() to your code:

```
from wheezy.html.ext.mako import widget_preprocessor

template_lookup = TemplateLookup(
        ...
        preprocessor=[widget_preprocessor])
```

The only thing widget_preprocessor() does is translation of widget code to adequate Mako code.

Let's demonstrate this with an example:

```
${model.remember_me.checkbox()}
```

is translated to the following Mako code (during template compilation phase):

```
<input id="remember-me" name="remember_me" type="checkbox" value="1"\
% if 'remember_me' in errors:
 class="error"\
% endif
% if model.remember_me:
 checked="checked"\
```

(continues on next page)

(continued from previous page)

```
% endif
 />
```

which effectively renders the HTML at runtime:

```
<input id="remember-me" name="remember_me" type="checkbox" value="1" />
```

Since widgets also decorate appropriate HTML tags in case of error, the `errors` dictionary must be available in the `Mako` context:

```
template = template_lookup.get_template(template_name)
assert 'errors' in kwargs
template.render(
            **kwargs
)
```

See *wheezy.html.ext.mako* for more examples.

Tenjin

wheezy.html integration with `Tenjin` is provided via the preprocessor feature. Here is how to add `widget_preprocessor()` to your code:

```
from wheezy.html.ext.tenjin import widget_preprocessor

engine = tenjin.Engine(
        ...
        pp=[widget_preprocessor])
```

The only thing `widget_preprocessor()` does is translation of widget code to adequate `Tenjin` code.

Let's demonstrate this with an example:

```
${model.remember_me.checkbox(class_='i')}
```

is translated to the following `Tenjin` code (during template compilation phase):

```
<input id="remember-me" name="remember_me" type="checkbox" value="1"<?py #pass ?>
<?py if 'remember_me' in errors: ?>
 class="error i"<?py #pass ?>
<?py else: ?>
 class="i"<?py #pass ?>
<?py #endif ?><?py if model.remember_me: ?>
 checked="checked"<?py #pass ?>
<?py #endif ?>
 />
```

which effectively renders the HTML at runtime:

```
<input id="remember-me" name="remember_me" type="checkbox" value="1" class="i" />
```

Since widgets also decorate appropriate HTML tags in case of error, the `errors` dictionary must be available in the `Tenjin` context:

```
assert 'errors' in kwargs
engine.render('page.html',
              **kwargs
)
```

See *wheezy.html.ext.tenjin* for more examples.

Wheezy Template

wheezy.html integration with `wheezy.template` is provided via the preprocessor feature. Here is how to add `WidgetExtension()` to your code:

```
from wheezy.html.ext.template import WidgetExtension
from wheezy.html.utils import html_escape
from wheezy.html.utils import format_value

engine = Engine(
         ...
         extensions=[
             WidgetExtension
])
engine.global_vars.update({
    'format_value': format_value,
    'h': html_escape
})
```

The only thing `WidgetExtension()` does is translation of widget code to adequate `wheezy.template` code.

Let's demonstrate this with an example:

```
@model.remember_me.checkbox(class_='i')
```

is translated to the following `wheezy.template` code (during template compilation phase):

```
<input id="remember-me" name="remember_me" type="checkbox" value="1"
@if 'remember_me' in errors:
 class="error i"
@else:
 class="i"
@if model.remember_me:
 checked="checked"
@end
 />
```

which effectively renders the HTML at runtime:

```
<input id="remember-me" name="remember_me" type="checkbox" value="1" class="i" />
```

Since widgets also decorate appropriate HTML tags in case of error, `errors` dictionary must be available in `wheezy.template` context:

```
assert 'errors' in kwargs
engine.render('page.html',
              **kwargs
)
```

See *wheezy.html.ext.template* for more examples.

2.4 Modules

2.4.1 wheezy.html

2.4.2 wheezy.html.utils

`utils` module.

`wheezy.html.utils.`**`date_format_provider`**(*value*, *format_spec=None*)

Default format provider for `datetime.date`.

Requires year >= 1900, otherwise returns an empty string.

```
>>> date_format_provider(date.min)
''
>>> date_format_provider(min_date)
'1900/01/01'
>>> date_format_provider(date(2012, 2, 6))
'2012/02/06'
```

`wheezy.html.utils.`**`datetime_format_provider`**(*value*, *format_spec=None*)

Default format provider for `datetime.datetime`.

Requires year >= 1900, otherwise returns an empty string.

```
>>> datetime_format_provider(datetime.min)
''
>>> datetime_format_provider(min_datetime)
'1900/01/01 00:00'
>>> datetime_format_provider(datetime(2012, 2, 6, 15, 17))
'2012/02/06 15:17'
```

`wheezy.html.utils.`**`escape_html`**(*s*)

Escapes a string so it is valid within HTML. Converts *None* to an empty string. Raises TypeError is *s* is not a string or unicode object.

```
>>> html_escape(None)
''
```

```
>>> escape_html('&<>"\'')
"&amp;&lt;&gt;&quot;'"
```

`wheezy.html.utils.`**`escape_html_native`**(*s*)

Escapes a string so it is valid within HTML. Converts *None* to an empty string. Raises TypeError is *s* is not a string or unicode object.

```
>>> html_escape(None)
''
```

```
>>> escape_html('&<>"\'')
"&amp;&lt;&gt;&quot;'"
```

`wheezy.html.utils.`**`format_value`**(*value*, *format_spec=None*, *format_provider=None*)

Formats widget value.

`format_provider` - a callable of the following form:

```
def my_formatter(value, format_spec):
    return value_formatted
```

```
>>> str(format_value(date(2012, 2, 6), '%m-%d-%y'))
'02-06-12'
>>> format_value(date(2012, 2, 6),
...         format_provider=lambda value, ignore:
...         value.strftime('%m-%d-%y'))
'02-06-12'
>>> list(map(str, format_value([1, 2, 7])))
['1', '2', '7']
>>> format_value([])
()
```

If format provider is unknown apply str.

```
>>> str(format_value({}))
'{}'
```

wheezy.html.utils.**html_escape**(*s*)

Escapes a string so it is valid within HTML. Converts *None* to an empty string. Raises TypeError is *s* is not a string or unicode object.

```
>>> html_escape(None)
''
```

```
>>> escape_html('&<>"\'')
"&amp;&lt;&gt;&quot;'"
```

2.4.3 wheezy.html.ext.lexer

lexer module

class wheezy.html.ext.lexer.**InlinePreprocessor**(*pattern*, *directories*, *strategy=None*)

Inline preprocessor

class wheezy.html.ext.lexer.**Preprocessor**(*widgets_pattern*)

Generic widget preprocessor.

checkbox(*expr*, *params*, *expr_filter*)

HTML element input of type checkbox.

dropdown(*expr*, *params*, *expr_filter*)

HTML element select.

emptybox(*expr*, *params*, *expr_filter*)

HTML element input of type text. Value is rendered only if evaluated to boolean True.

error(*expr*, *params*, *expr_filter*)

General error message or field error.

error_class(*name*, *class_*)

Checks for error and add css class error.

expression(*text*, *expr_filter=''*)

Interpretate text as string expression or python expression.

hidden (*expr*, *params*, *expr_filter*)
HTML element input hidden.

info (*expr*, *params*, *expr_filter*)
General info message.

input_helper (*expr*, *params*, *expr_filter*, *input_type*)
HTML element input of type input_type.

join_attrs (*kwargs*)
Joins `kwargs` as html attributes.

label (*expr*, *params*, *expr_filter*)
HTML element label.

listbox (*expr*, *params*, *expr_filter*)
HTML element select of type multiple.

message_helper (*expr*, *params*, *expr_filter*, *msg_class*)
General info message.

multiple_checkbox (*expr*, *params*, *expr_filter*)
Multiple HTML element input of type checkbox.

multiple_hidden (*expr*, *params*, *expr_filter*)
Multiple HTML element input of type hidden.

password (*expr*, *params*, *expr_filter*)
HTML element input of type password. Value is rendered only if it is not None or ‘’.

radio (*expr*, *params*, *expr_filter*)
A group of HTML input elements of type radio.

textarea (*expr*, *params*, *expr_filter*)
HTML element textarea.

textbox (*expr*, *params*, *expr_filter*)
HTML element input of type text. Value is rendered only if it is not None or ‘’.

warning (*expr*, *params*, *expr_filter*)
General warning message.

class `wheezy.html.ext.lexer.`**WhitespacePreprocessor** (*rules*, *ignore_rules=None*)
Whitespace preprocessor.

2.4.4 wheezy.html.ext.parser

`parser` module

`wheezy.html.ext.parser.`**parse_args** (*text*)
Parses argument type of parameters.

```
>>> parse_args('')
[]
>>> parse_args('10, "x"')
['10', '"x"']
>>> parse_args("'x', 100")
["'x'", '100']
>>> parse_args('"Account Type:"')
['"Account Type:"']
```

wheezy.html.ext.parser.**parse_known_function**(*expr*)

Parses known functions.

```
>>> parse_known_function("dob")
('dob', 'dob')
>>> parse_known_function("dob.format()")
('dob', 'format_value(dob, None)')
>>> parse_known_function("user.dob.format(_('YYYY/MM/DD'))")
('user.dob', "format_value(user.dob, _('YYYY/MM/DD'))")
>>> parse_known_function("user.dob.format(format_provider=lambda value, ignore:␣
↪value.strftime('%m-%d-%y'))")
('user.dob', "format_value(user.dob, format_provider=lambda value, ignore: value.
↪strftime('%m-%d-%y'))")
```

wheezy.html.ext.parser.**parse_kwargs**(*text*)

Parses key-value type of parameters.

```
>>> parse_kwargs('choices=account_types')
{'choices': 'account_types'}
>>> sorted(parse_kwargs('autocomplete="off", maxlength=12').items())
[('autocomplete', '"off"'), ('maxlength', '12')]
```

wheezy.html.ext.parser.**parse_name**(*expr*)

Parses name from expression of the following form:

```
[object.]name[.format(...]
```

```
>>> parse_name('display_name')
'display_name'
>>> parse_name('account.display_name')
'display_name'
>>> parse_name('account.display_name.format(')
'display_name'
```

wheezy.html.ext.parser.**parse_params**(*text*)

Parses function parameters.

```
>>> parse_params('')
([], {})
>>> parse_params('choices=account_types')
([], {'choices': 'account_types'})
>>> parse_params('"Account Type:"')
(['"Account Type:"'], {})
>>> parse_params('"Account Type:", class_="inline"')
(['"Account Type:"'], {'class': '"inline"'})
```

wheezy.html.ext.parser.**parse_str_or_int**(*text*)

Interpretate `text` as string or int expression.

```
>>> parse_str_or_int('"Hello"')
'Hello'
>>> parse_str_or_int("'Hello'")
'Hello'
>>> parse_str_or_int('100')
'100'
>>> parse_str_or_int('model.username')
```

2.4.5 wheezy.html.ext.jinja2

`jinja2` extension module.

class `wheezy.html.ext.jinja2.`**InlineExtension**(*searchpath*, *fallback=False*)

Inline preprocessor. Rewrite {% inline "..." %} tag with file content. If fallback is `True` rewrite to {% include "..." %} tag.

```
>>> t = '1 {% inline "master.html" %} 2'
>>> m = RE_INLINE.search(t)
>>> m.group('path')
'master.html'
>>> t[:m.start()], t[m.end():]
('1 ', ' 2')
>>> m = RE_INLINE.search(' {% inline "shared/footer.html" %}')
>>> m.group('path')
'shared/footer.html'
```

preprocess(*source*, *name*, *filename=None*)

This method is called before the actual lexing and can be used to preprocess the source. The *filename* is optional. The return value must be the preprocessed source.

class `wheezy.html.ext.jinja2.`**Jinja2Preprocessor**(*variable_start_string=None*, *variable_end_string=None*)

class `wheezy.html.ext.jinja2.`**WhitespaceExtension**(*environment*)

preprocess(*source*, *name*, *filename=None*)

This method is called before the actual lexing and can be used to preprocess the source. The *filename* is optional. The return value must be the preprocessed source.

class `wheezy.html.ext.jinja2.`**WidgetExtension**(*environment*)

preprocess(*source*, *name*, *filename=None*)

This method is called before the actual lexing and can be used to preprocess the source. The *filename* is optional. The return value must be the preprocessed source.

2.4.6 wheezy.html.ext.mako

`mako` extension module.

class `wheezy.html.ext.mako.`**MakoPreprocessor**(*skip_imports=False*)

`wheezy.html.ext.mako.`**inline_preprocessor**(*directories*, *fallback=False*)

Inline preprocessor. Rewrite <%inline file="..." /> tag with file content. If fallback is `True` rewrite to <%include file="..." /> tag.

```
>>> t = '1 <%inline file="master.html"/> 2'
>>> m = RE_INLINE.search(t)
>>> m.group('path')
'master.html'
>>> t[:m.start()], t[m.end():]
('1 ', ' 2')
>>> m = RE_INLINE.search(' <%inline file="shared/footer.html"/>')
>>> m.group('path')
'shared/footer.html'
```

2.4.7 wheezy.html.ext.template

`wheezy.template` extension module.

class `wheezy.html.ext.template.`**InlineExtension**(*searchpath*, *fallback=False*)

Inline preprocessor. Rewrite @inline("...") tag with file content. If fallback is `True` rewrite to @include("...") tag.

```
>>> t = '1 @inline("master.html") 2'
>>> m = RE_INLINE.search(t)
>>> m.group('path')
'master.html'
>>> t[:m.start()], t[m.end():]
('1 ', ' 2')
>>> m = RE_INLINE.search(' @inline("shared/footer.html")')
>>> m.group('path')
'shared/footer.html'
```

class `wheezy.html.ext.template.`**WheezyPreprocessor**

2.4.8 wheezy.html.ext.tenjin

`tenjin` extension module.

class `wheezy.html.ext.tenjin.`**TenjinPreprocessor**

`wheezy.html.ext.tenjin.`**inline_preprocessor**(*directories*, *fallback=False*)

Inline preprocessor. Rewrite <?py inline("...") ?> tag with file content. If fallback is `True` rewrite to <?py include("...") ?> tag.

```
>>> t = '1 <?py inline("master.html") ?> 2'
>>> m = RE_INLINE.search(t)
>>> m.group('path')
'master.html'
>>> t[:m.start()], t[m.end():]
('1 ', ' 2')
>>> m = RE_INLINE.search(' <?py inline("shared/footer.html") ?>')
>>> m.group('path')
'shared/footer.html'
```

Python Module Index

W

Index

R

T

W