Flask-Genshi Documentation

Release 0.4

Dag Odenhall

September 14, 2011

CONTENTS

Flask-Genshi is an extension to Flask that allows you to easily use Genshi for templating.

Features:

- Integrates fully with Flask and works with Flask-Babel for internationalization and flatland for form processing (that is, the special support flatland has for Genshi). Other extensions such as Flask-WTF and Flask-Creole should work fine too.
- Selects configurable rendering methods from the template filename, taking care of the mimetype, doctype and XML serialization, i.e., how to close tags et cetera.

Not yet:

- Flask-Theme probably does not work with Genshi templates.
- Chameleon, not sure if it should be a separate extension or not.

CHAPTER

ONE

WHY GENSHI?

Genshi is one of the slower templating toolkits, the idea of writing XML for templating seem horrifying and Jinja is a very nice text-based template engine. Why would Genshi possibly be interesting?

- Templates are rarely the culprit in application performance. If Genshi does become a culprit you can render your templates with Chameleon which performs very well.
- XML lends itself naturally to templating HTML. Genshi templates are usually much more terse and readable because Python constructs can be terminated by the XML end tags and translation strings can be extracted directly from the content of text elements.

  We can also operate programmatically on template streams and run filters and transformations. We can have a layout template extract a subtitle from a page template's title tag. Many possibilities.

  Take this Jinja template:

  ```
  {% block title %}Page Title{% endblock %}
  {% if show_paragraph %}
    <p>{% trans %}Example paragraph with {{ variable }} content{% endtrans %}
  {% endif %}
  ```

  Compare with the Genshi version:

  ```
  <title>Page Title</title>
  <p py:if="show_paragraph">Example paragraph with $variable content</p>
  ```

- For better or worse, Python in Genshi templates is really Python. The only difference is that, like with Jinja, attribute access falls back on item access and vice versa. You need only know some basic Python and XML to write Genshi templates, and you can use existing XML tools such as support in editors.
- Because invalid XML results in template errors, valid output is enforced without a need to use HTML validators. You get the benefits of XHTML with browser-friendly output, and you can swap the output format on-the-fly. You're also writing actual XML so no need for those pesky spaces in self-closing tags: `<hr/>` will render to HTML as `<hr>` and to XHTML as `<hr />`.

CHAPTER

TWO

INSTALLATION

Just grab it from PyPI with *easy_install* or *pip*, for example:

```
$ easy_install Flask-Genshi
```

If you're starting a new project you don't need to explicitly install Flask as Flask-Genshi depends on it already.

CHAPTER

THREE

HOW TO USE

You need to construct a `Genshi` with your `Flask` instance.

```
from flaskext.genshi import Genshi

app = Flask(__name__)
genshi = Genshi(app)
```

The best way to render templates is to use `render_response()`. This ensures that the proper mimetype is sent if you render XHTML, XML or text, and sets the right doctype for you.

Use it like so:

```
from flaskext.genshi import render_response

@app.route('/')
def index():
    title = 'Genshi + Flask, a match made in heaven!'
    return render_response('index.html', dict(title=title))
```

CHAPTER

FOUR

USING METHODS

Methods control things such as the doctype and how end tags are rendered, and with `render_response()` also the mimetype. Unless overridden the method used is decided by the template's filename extension.

By default HTML renders as strict HTML 4.01. This is how you change it to HTML5:

```
genshi.extensions['html'] = 'html5'
```

I recommend against this but of course you can also change it to XHTML:

```
genshi.extensions['html'] = 'xhtml'
```

You can also override the default with a parameter to the templating functions:

```
render_response('video.html', method='html5')
```

You can add and edit any methods and extensions you like, here are the defaults:

Method	Extension	Template Language	Output
`'html'`	*.html*	XML based	HTML 4.01 Strict
`'html5'`		XML based	HTML 5
`'xhtml'`		XML based	XHTML 1.0 Strict
`'xml'`	*.xml*	XML based	*application/xml*
`'svg'`	*.svg*	XML based	SVG 1.1
`'text'`	*.txt*	Text based	*text/plain*
`'js'`	*.js*	Text based	*application/javascript*
`'css'`	*.css*	Text based	*text/css*

CHAPTER

FIVE

USING JINJA FILTERS AND TESTS

Flask-Genshi supports tests and filters from Jinja:

```
<p class="${'even' if tests.even(1) else 'odd'}">
  ${filters.truncate('Hello World', 10)}
</p>
```

Result:

```
<p class="odd">
  Hello ...
</p>
```

CHAPTER

SIX

MODULE TEMPLATES

Flask can load templates specific to a `flask.Module` and Flask-Genshi also supports this. It works just like in Flask:

```
render_response('modname/template.html')
```

This will first look for, e.g. `app/templates/modname/template.html`, and if not found, e.g. `app/mods/modname/templates/template.html`. This lets you override module templates in your application. New in version 0.4.

CHAPTER

SEVEN

CONTEXT PROCESSORS

Context processors work as expected.

```
@app.context_processor
def add_site_name():
    return dict(site_name=app.config['SITE_NAME'])
```

This lets you use `site_name` in templates without including it in every render call.

```
<title>$site_name</title>
```

New in version 0.4.

CHAPTER

EIGHT

RENDER FROM STRINGS

Like `flask.render_template_string()`, Flask-Genshi lets you render strings as templates:

```
render_response(string='Hello $name', context={'name': 'World'}, method='text')
```

The same pattern applies to all functions. New in version 0.4.

CHAPTER

NINE

USING WITH FLATLAND

First, add the flatland filter:

```
from flatland.out.genshi import flatland_filter

@genshi.filter('html')
def inject_flatland(template, context):
    return flatland_filter(template, context)
```

Don't forget the form namespace:

```
<html xmlns:form="http://ns.discorporate.us/flatland/genshi">
  <input type="text" form:bind="form.username"/>
</html>
```

New in version 0.5.

CHAPTER

TEN

USING WITH FLASK-BABEL

You can use Flask-Genshi with Flask-Babel for internationalization. First, set up the `Translator` filter with the callback interface via `template_parsed()`:

```
from genshi.filters import Translator
from flaskext.babel import gettext

@genshi.template_parsed
def callback(template):
    Translator(gettext).setup(template)
```

You'll want a `babel.cfg` similar to this one:

```
[python: **.py]
[genshi:**/templates/**.html]
[genshi:**/templates/**.txt]
template_class = genshi.template.NewTextTemplate
```

Consult the Genshi documentation on Internationalization and Localization for details on extracting translation strings from Genshi templates. New in version 0.5.

CHAPTER

ELEVEN

SIGNAL SUPPORT

Flask-Genshi supports a signal similar to `flask.template_rendered` called `template_generated`. It is sent the template object and the context that was used to generate the template stream, when one is successfully generated. New in version 0.5.

CHAPTER

TWELVE

FILTERING TEMPLATE STREAMS

You can apply filters to templates that you render globally per rendering method:

```
from genshi.filters import Transformer

@genshi.filter('html')
def prepend_title(template):
    return template | Transformer('head/title').prepend('MySite - ')
```

Alternatively, since version 0.5, you can apply a filter for a template as you render it, more repetitive but with more control:

```
render_response('index.html', filter=prepend_title)
```

See the Genshi documentation for more filters you can use. New in version 0.3.Changed in version 0.5: Filters can now optionally take a second argument for the context.Changed in version 0.5: You can now apply filters on a per-render basis.

CHAPTER

THIRTEEN

API REFERENCE

class flaskext.genshi.**Genshi**(*app=None*)

Initialize extension.

```
app = Flask(__name__)
genshi = Genshi(app)
```

Changed in version 0.4: You can now initialize your application later with init_app().Deprecated since version 0.4: app.genshi_instance in favor of app.extensions['genshi'].

callback

A callable for Genshi's callback interface, called when a template is loaded, with the template as the only argument.

template_parsed() is a decorator for setting this. New in version 0.5.

extensions

What method is used for an extension.

filter(**methods*)

Decorator that adds a function to apply filters to templates by rendering method. New in version 0.3.Changed in version 0.5: Filters can now optionally take a second argument for the context.

filters

Filter functions to be applied to templates. New in version 0.3.

init_app(*app*)

Initialize a Flask application for use with this extension. Useful for the factory pattern but not needed if you passed your application to the Genshi constructor.

```
genshi = Genshi()

app = Flask(__name__)
genshi.init_app(app)
```

New in version 0.4.

methods

Render methods. Changed in version 0.3: Support for Javascript and CSS.Changed in version 0.4: Support for SVG.

template_loader

A genshi.template.TemplateLoader that loads templates from the same places as Flask.

template_parsed(*callback*)

Set up a calback to be called with a template when it is first loaded and parsed. This is the correct way to set up the Translator filter. New in version 0.5.

flaskext.genshi.**template_generated**
Signal emitted when a template stream has been successfully generated, passing *template* and *context* via the app as sender. New in version 0.5.

flaskext.genshi.**generate_template**(*template=None*, *context=None*, *method=None*, *string=None*, *filter=None*)
Creates a Genshi template stream that you can run filters and transformations on.

flaskext.genshi.**render_template**(*template=None*, *context=None*, *method=None*, *string=None*, *filter=None*)
Renders a template to a string.

flaskext.genshi.**render_response**(*template=None*, *context=None*, *method=None*, *string=None*, *filter=None*)
Renders a template and wraps it in a `response_class` with mimetype set according to the rendering method.

PYTHON MODULE INDEX

f