pail Documentation

Release 0.1

Uli Fouquet

May 07, 2013

CONTENTS

pail is a WSGI middleware providing Adaptive Images. Delivering small images to small devices.

It detects your visitor's screen size and automatically creates, and delivers device appropriate re-scaled versions of your web page's embeded HTML images. No (major) mark-up changes needed. It is intended for use with Responsive Designs and to be combined with Fluid Image techniques.

This package is based on the ideas of Matt Wilcox and (more loosely) on his PHP script for the same purpose. Matt is in no way to blame for any shortcomings of this Python port.

pail provides special support for use with Paste.

Please note, that this package is still in a very early state and changes, also to the API, are likely to happen in near future.

Comments and patches are welcome. Please send these to uli at gnufix dot de.

CHAPTER

ONE

INSTALLATION

The package can be installed by:

```
$ pip install pail
```

Afterwards you should be able to use pail in any WSGI environment. See the documentation for details.

CHAPTER

TWO

LINKS

- Full documentation (including deployment examples)
- Fork me on GitHub

CHAPTER

THREE

DOCUMENTATION (DETAILED)

3.1 How it works

pail does a number of things depending on the scenario, the component has to handle but here's a basic overview of what happens when you load a page with *pail* enabled on the server:

1. The HTML starts to load in the browser and a snippet of JS in the `<head>` writes a cookie, storing the visitor's screen size in pixels.
2. The browser then encounters an `<img>` tag and sends a request to the server for that image. It also sends the cookie, because that's how browsers work.
3. The web server sends the incoming request to the local WSGI app with *pail* acting as a filter somewhere in the pipeline of WSGI apps.
4. *pail* feeds any wrapped WSGI application (the real content provider) and receives some HTTP response from that application which also includes the image requested by the visitor.
5. *pail* looks for a cookie and finds, that the user has a maximum screen size of 480px.
6. It compares the cookie value with all `resolutions` that were configured, and decides which matches best. In this case, an image maxing out at 480px wide.
7. It checks the image width. If that's smaller than the user's screen width it sends the image unchanged.
8. If it is larger, *pail* creates a down-scaled copy and sends it to the user.

pail also does a few other things when needs arise, for example:

- Detects Retina displays. You can choose to serve high DPI images to those displays if you want, by using an alternate line of JavaScript.
- It handles cases where there isn't a cookie set; mobile devices will be supplied the smallest image, and desktop devices will get the largest.

There are also some things, *pail* does **not** support currently (and which the original PHP script does):

- There is no caching yet.
- Cache headers are not set correctly yet.

3.2 Deployment – How to Use *pail*

pail provides a WSGI middleware component

```
from pail.wsgi import ImageAdaptingMiddleware
```

that does all the work. It requires a list of supported *resolutions* (string with a comma-separated list of integers like `"480, 922, 1322"`). Also a factory for this middleware is available:

```
from pail.wsgi import filter_app
```

that returns instances of the middleware. See the *API* for details.

3.2.1 With Paste

pail provides a Paste-compatible WSGI filter app named `main` (this means that Paste, once *pail* is installed, can find the middleware as `egg:pail` automatically). It is meant to be used as a WSGI application wrapper (a 'filter app' in Paste terms), so that incoming requests are parsed, then passed to the wrapped application and the result is scanned for image data before it is passed on to the client.

A simple Paste config that makes use of *pail* might look like this:

```
[server:main]
use = egg:Paste#http
host = 0.0.0.0
port = 8000

[filter-app:main]
use = egg:pail
resolutions = 1024, 480
next = static

[app:static]
use = egg:Paste#static
document_root = %(here)s/static-dir/
```

Here the first section `[server:main]` defines a server listening on port 8000.

The `[app:static]` section tells where the real content comes from: it's simple static content read from local dir *static-dir*. We use the Paste app *static* for that purpose.

The relevant part, however, is the second section `[filter-app:main]`, where we tell Paste to filter the static content through the *pail* filter app: `use = egg:pail`.

Then we tell the *pail* filter to support the two resolutions of *1024* and *480*: `resolutions = 1024, 480`.

Finally we state to go on with the static app *static*: `next = static`. With Paste each filter app needs some other app (or filter app) to filter.

Of course, this is only a very plain sample for a WSGI/Paste setup. You could also create much more complex pipelines with several other filters and adapting images from Plone, Diazo or other content providers.

Also the content produced by *pail* could be mangled by further filters in a WSGI pipeline; that's up to you. See the respective *paste.deploy* documentation for details about Paste configuration files.

Example

Create a virtual environment, activate it, and install pail:

```
$ virtualenv py27   # also Python 2.6, 3.2, 3.3 should work
$ source py27/bin/activate
(py27)$ pip install pail
```

Create a Paste config in `pailstatic.ini`:

```
# pailstatic.ini
#
[server:main]
# run an HTTP server on port 8000
use = egg:Paste#http
host = 0.0.0.0
port = 8000

[filter-app:main]
# filter all requests through pail
use = egg:pail
resolutions = 1024, 480
next = static

[app:static]
# serve static content...
use = egg:Paste#static
# ...from this local directory
document_root = %(here)s/static/
```

Now create the static content:

```
(py27)$ mkdir static/
(py27)$ cd static/
```

Create an HTML file named `index.html` like this:

```
<html>
  <head>
    <title>My test page</title>
    <script>
      document.cookie='resolution='+Math.max(
                      screen.width,screen.height)+'; path=/';
    </script>
  </head>
  <body>
    <div>Some Text</div>
    <img  style="width: 100%;" href="myimage.jpg" />
  </bod>
</html>
```

and copy some image file, preferably a wide one (1024+ pixels width), into the `static/` dir. Rename the image file to `myimage.jpg`.

Now install the missing `paster` packages and start the server:

```
(py27)$ pip install PasteScript
(py27)$ paster serve pailstatic.ini
```

Now, browsing http://localhost:8000/ you should see the generated page with the image included. Nothing special. Nothing? If you are using a desktop and the original image (put into the static dir) was wider than 1024 pixels, while the desktop has a maximum resolution of 1024px, it should automatically have been *downscaled to 1024 px* width. The same page watched from a mobile device with <= 480 px screen width should automatically get an image with *width 480 px*.

You can force that switch on a single machine by replacing the `Math.max()` expression in the JavaScript part to some fixed value like `480`, `960`, or whatever you want.

3.3 Developers' Instructions

These are instructions for developers that want to develop *pail* itself - not for users or webmasters.

Before starting any further work it is highly recommended to create and activate a virtualenv:

```
$ virtualenv py27
$ source py27/bin/activate
(py27)$
```

Here we used a Python 2.7 install but *pail* is also tested with Python 2.6, 3.2, and 3.3.

Now get the source via GitHub

```
$ git clone https://github.com/ulif/pail
```

and change into the created *pail/* directory.

The development setup is done with:

```
(py27)$ python setup.py dev
```

This step mainly installs the required external packages (mainly *Pillow* and *WebOb*) and needed testing components (*py.test* and *pytest-cov*) locally in your virtualenv.

3.3.1 Testing

pail testing uses *py.test*. The recommended way to run tests is therefore:

```
(py27)$ py.test
```

py.test should be installed already if you completed the steps above.

You could also run `$ python setup.py tests`, but this approach is less flexible. For instance you currently cannot pass arguments to the test-runner.

For testing with several Python versions in one row *pail* also provides a `tox.ini`. So, if you have tox installed, you can run tests for different Python versions like this:

```
(py27)$ pip install tox  # required only once
(py27)$ tox
```

Modify *tox.ini* to your needs.

3.3.2 Test-Coverage

A coverage report can also be created with:

```
(py27)$ py.test --cov=pail --cov-report=html
```

Results can be found in *htmlcov/* afterwards. Before submitting patches please make sure that test coverage is at 100%.

3.3.3 Spinx-Docs

The *pail* docs are created using Sphinx. The required packages can be installed locally doing:

```
(py27)$ python setup.py docs
```

This will not generate the docs but install the packages needed to create the docs, most notably Sphinx.

The actual docs can then be created with:

```
(py27)$ sphinx-build docs/ docs/_build/html
```

Sources for the docs can be found (you guessed it) in the `docs/` directory.

3.3.4 Running the WSGI Middleware Locally

If you want to see the whole machinery in real action, you need some local (WSGI) server. Using Paste this is not difficult to set up.

See *Deployment – How to Use pail* for details.

3.4 API

3.4.1 WSGI Components

Middleware and other components that can be used with other WSGI components.

class `pail.wsgi.`**`ImageAdaptingMiddleware`**(*app*, *global_conf*, *resolutions='1382*, *992*, *768*, *480'*)
Bases: `object`

A WSGI middleware to shrink images on-the-fly.

It works as a filter-wrapper that examines images delivered by other WSGI components and might reduce the image sizes on-the-fly.

`acceptable_types` = **['image/jpeg', 'image/png', 'image/gif']**
The content types considered as handable. Only HTTP responses providing one of these types are handled by the middleware.

`create_resized_image`(*response*, *resolution*)
Create a resized version of current content image.

`get_client_resolution`(*request*)
Get the client screen resolution from request.

`get_resolution`(*request*)
Determine a desired resolution from client screen resolution and avaiilable resolutions.

`should_adapt`(*response*)
Should we adapt the response from the wrapped app?

`should_ignore`(*request*, *resolution*)
Should the given request be ignored?

`pail.wsgi.`**`filter_app`**(*app*, *global_conf*, ***kw*)
A factory that returns *ImageAdaptingMiddleware* instances.

3.4.2 Helper Functions

Minor helper functions.

pail.helpers.**get_file_length**(*fd*)
Get length of file denoted by a file descriptor.

As we cannot get a stat of files of which we only have a file descriptor (and no path or similar), we have to do some tricks to get the file-length anyway.

fd should be the descriptor of a file already open for reading.

Returns the length of file as an integer.

pail.helpers.**get_resolution**(*client_width*, *pixel_density*, *resolution_list*, *is_mobile=True*)
Get the desired resolution based on the input parameters.

client_width The screen width of some client device.

pixel_density By default 1. The amount of physical pixels representing a single CSS pixel. Normally 1 but on retina displays it can be more.

resolution_list A list of supported screen widths (integers) from the web applications side.

is_mobile A boolean. Indicates, whether the client device seems to be a mobile device. If so and no *client_width* is given, then the lowest possible resolution (based on *resolution_list*) is returned.

pail.helpers.**resize**(*image*, *resolution=None*, *client_resolution=None*, *enlarge=False*)
Resize *image* to have width *resolution*.

image can be a path or some open file descriptor.

resolution gives the desired maximum width in pixels.

Returns `None` or a tuple

```
(<FORMAT>, <RESULT_FILE_FD>)
```

with `<FORMAT>` being an image type as determined by PIL and `<RESULT_FILE_FD>` being an opened temporary file.

If the image is already narrower than *resolution*, *None* is returned.

The same applies if the image cannot be read or the resizing operation fails.

pail.helpers.**to_int_list**(*string*)
Try to turn a string into a list of integers.

Turns strings like *'1, 2, 3'* into regular lists like *[1, 2, 3]*. The number have to be comma-separated. Also lists with only one entry are handled correctly.

CHAPTER

FOUR

CREDITS

Adapting Images base idea: Matt Wilcox

Created by Uli Fouquet

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

CHAPTER

SIX

LICENSE

CHAPTER

SEVEN

CHANGES

7.1 0.1 (2012-05-07)

- First implementation based on Matt Wilcox' PHP script.

PYTHON MODULE INDEX

p

pail.helpers, **??**
pail.wsgi, **??**