# akhet Documentation

***Release 2.0***

**Mike Orr**

August 08, 2014

Contents

**Version** 2.0, released 2012-02-12

**Docs-Updated** same

**PyPI** http://pypi.python.org/pypi/Akhet

**Docs** http://readthedocs.org/docs/akhet/en/latest/

**Source** https://github.com/Pylons/akhet

**Bugs** https://github.com/Pylons/akhet/issues

**Discuss** pylons-discuss list

**Author** Mike Orr

**Contributors** Michael Merickel, Marcin Lulek

Akhet is a Pyramid library and demo application with a Pylons-like feel.

**Main changes in version 2:** **(A)** The 'akhet' scaffold gone, replaced by a demo application, which you can cut and paste from. **(B)** General Pyramid/Pylons material has been moved out of the manual to the Pyramid Cookbook, section Pyramid for Pylons Users guide. *(The guide is not yet online as of February 2012.)* **(C)** The include for static routes has changed to "akhet.static", but "akhet" is still allowed for backward compatibility. **(D)** A new pony module. **(E)** The repository is now on GitHub in the Pylons Project.

The demo is distributed separately from the Akhet. Its repository URL is in the Demo section.

Akhet runs on Python 2.5 - 2.7. Version 2 has been tested on Pyramid 1.3a6 and 1.2.4 using Pyramid 2.7.2 on Ubuntu Linux 11.10. The next Akhet version, 2.1, will focus on Python 3 and will drop Python 2.5. The demo application currently has the same compatibility range as Akhet itself.

The word "akhet" is the name of the hieroglyph that is Pylons' icon: a sun shining over two pylons. It means "horizon" or "mountain of light".

CHAPTER 1

# Documentation Contents

## 1.1 Library

### 1.1.1 Static route

The static route helper provides a more Pylons-like way to serve static files compared to Pyramid's standard static view. In Pylons, the static app is an overlay on "/", so that it looks first for a file under the static directory, and if no file exists, it falls back to the dynamic application. This static route helper works the same way: it works as an overlay on "/", so the route matches only if the file exists, otherwise Pyramid falls through to the next route. The difference is that Pylons' static app is a WSGI middleware, while the static route helper registers an ordinary route and a view. By convention you put the static route last in the route list, but it can go anywhere in the list.

Pyramid's standard static view, in contrast, works only with a URL prefix like "/static"; it can't serve top-level URLs like "/robots.txt" and "/favicon.ico". If you want a separate disjunct prefix like "/w3c" (for "/w3c/p3p.xml", the Internet standard for a machine-readable privacy policy), you'd have to configure a separate view and static directory for that prefix. With the static route helper you don't have to configure anything extra, just create a file "myapp/static/w3c/p3p.xml" and you're done.

The static route helper does have some disadvantages compared to Pyramid's static view. (1) There's no spiffy method to generate URLs to them. (2) You can't switch to a static media server and configure the nonexistent spiffy method to generate external URLs to it. (3) You can't override assets.

For completeness, we'll mention that if you're using Pyramid's static view, there are a couple workarounds for serving top-level URLs or disjoint URLs. (1) Use an ordinary route and view to serve a static file. (2) Use the "pyramid_assetviews" package on PyPI to serve top-level files. So you can weigh these alternatives against the static route helper. I (the static route author) am now undecided, so I can't definitively say which way is better. The demo app uses it mainly so that you can see it in action.

#### Usage

```
# In main().
config.include("akhet.static")
config.add_static_route("myapp", "static")
```

#### API

config.**add_static_route**(*package*, *subdir*, *cache_max_age=3600*, ***add_route_args*)

Register a route and view to serve any URL if a corresponding file exists under the static directory. If the file doesn't exist, the route will fail and Pyramid will continue down the route list.

Arguments:

- `package`: the name of the Python package containing the static files.
- `subdir`: the subdirectory within the package that contains the files. This should be a relative directory with "/" separators regardless of platform.
- `cache_max_age`: Influences the "Expires" and "Max-Age" HTTP headers in the response. (Default is 3600 seconds = 5 minutes.)
- `add_route_args`: Additional arguments for `config.add_route`. `name` defaults to "static" but can be overridden. (Every route in your application must have a unique name.) `pattern` and `view` may not be specified; it will raise TypeError if they are.

The API is from Pyramid's early days, so it makes an asset spec out of `package` and `subdir` for you and doesn't allow you to supply your own. It also searches only a single directory rather than a search path. These limitations may be relaxed in a future version.

#### Changes in version 2

The include module is now "akhet.static"; in version 1 it was "akhet". A backward compatibility shim is in place.

### 1.1.2 URL generator

A class that consolidates Pyramid's various URL-generating functions into one concise API that's convenient for templates. It performs the same job as `pylons.url` in Pylons applications, but the API is different.

Pyramid has several URL-generation routines but they're scattered between Pyramid request methods, WebOb request methods, Pyramid request attributes, WebOb request attributes, and Pyramid functions. They're named inconsistently, and the names are too long to put repeatedly in templates. The methods are usually – but not always – paired: one method returning the URL path only ("/help"), the other returning the absolute URL ("http://example.com/help"). Pylons defaults to URL paths, while Pyramid tends to absolute URLs (because that's what the methods with "url" in their names return). The Akhet author prefers path URLs because the automatically adjust under reverse proxies, where the application has the wrong notion of what its visible scheme/host/port is, but the browser knows which scheme/host/port it requested the page on.

`URLGenerator` unifies all these by giving short one-word names to the most common methods, and having a switchable default between path URLs and absolute URLs.

#### Usage

Copy the "subscribers" module in the Akhet demo (*akhet_demo/subscribers.py*) to your own application, and modify it if desired. Then, include it in your main function:

```
# In main().
config.include(".subscribers")
```

The subscribers attach the URL generator to the request as `request.url_generator`, and inject it into the template namespace as `url`.

`URLGenerator` was contributed by Michael Merickel and modified by Mike Orr.

#### API

**class** `akhet.urlgenerator.`**`URLGenerator`**(*context*, *request*, *qualified=False*)

**__call__**(**elements*, ***kw*)
Same as the `.route` method.

**__init__**(*context*, *request*, *qualified=False*)
Instantiate a URLGenerator based on the current request.

- `request`: a Pyramid Request.
- `context`: a Pyramid Context.
- `qualified`: If true, return fully-qualified URLs with the "scheme://host" prefix. If false (default), return only the URL path if the underlying Pyramid function allows it.

**app**
The application URL or path.

I'm a "reified" attribute which means I start out as a property but I turn into an ordinary string attribute on the first access. This saves CPU cycles if I'm accessed often.

I return the application prefix of the URL. Append a slash to get the home page URL, or additional path segments to get a sub-URL.

If the constructor arg 'qualified' is true, I return `request.application_url`, otherwise I return `request.script_name`.

**ctx**
The URL of the default view for the current context.

I'm a "reified" attribute which means I start out as a property but I turn into an ordinary string attribute on the first access. This saves CPU cycles if I'm accessed often.

I am mainly used with traversal. I am different from `.app` when using context factories. I always return a qualified URL regardless of the constructor's 'qualified' argument.

**current**(**elements*, ***kw*)
Generate a URL based on the current request's route.

I call `pyramid.url.current_route_url`. I'm the same as calling `.route` with the current route name. The result is always qualified regardless of the constructor's 'qualified' argument.

**resource**(**elements*, ***kw*)
Return a "resource URL" as used in traversal.

`*elements` is the same as with `.route`. Keyword args `query` and `anchor` are the same as the `_query` and `_anchor` args to `.route`.

When called without arguments, I return the same as `.ctx`.

**route**(*route_name*, **elements*, ***kw*)
Generate a route URL.

I return a URL based on a named route. Calling the URLGenerator instance is the same as calling me. If the constructor arg 'qualified' is true, I call `pyramid.url.route_url`, otherwise I call `pyramid.url.route_path`.

Arguments:

- `route_name`: the name of a route.
- `*elements`: additional segments to append to the URL path.

Keyword arguments are passed to the underlying function. The following are recognized:

- `_query`: the query parameters. May be a dict-like object with an `.items()` method or a sequence of 2-tuples.

- `_anchor`: the URL's "#ancor" fragment without the "#".
- `_qualified`: override the constructor's "qualified" flag.
- `_app_url`: override the "scheme://host" prefix. (This also causes the result to be qualified if it wouldn't otherwise be.)
- Other keyword args override path variables defined in the route.

If the relevant route has a *pregenerator* defined, it may modify the elements or keyword args.

#### Subclassing

The source code (*akhet/urlgenerator.py*) has some commented examples of things you can do in a subclass. For instance, you can define a `static` method to generate a URL to a static asset in your application, or a `deform` method to serve static files from the Deform form library. The instance has `request` and `context` attributes, which you can use to calculate any URL you wish. You can put a subclass in your application and then adjust the subscribers to use it.

The reason the base class does not define a `static` method, is that we're not sure yet what the best long-term API is. We want something concise enough for everyday use but also supporting unusual cases, and something we can guarantee is correct and we're comfortable supporting long-term. There's also the issue of the static route helper vs Pyramid's static view, or multiple Pyramid static views responding to different sub-URLs. In the meantime, if you want a `static` method, you can decide on your own favorite API and implement it.

### 1.1.3 Pony

`akhet.pony` is a port of `paste.pony` in the Paste distribution, originally written by Ian Bicking. Usage:

```
# In main().
config.include("akhet.pony")
```

This registers a route at URL "/pony", which displays an ASCII art pony. If the user appends the query parameter "horn" with a non-blank value, as in */pony?horn=1*, it will display a unicorn instead.

The page does not show your application name or anything in your site template, but it does include a "Home" hyperlink which returns to the application's home page (normally "/" unless the application is mounted under a URL prefix).

## 1.2 Demo application

### 1.2.1 Usage and features

The Akhet demo application shows the Akhet library's features in action, and contains templates and code you can copy into your own application as a starting point. The demo is based on the former Akhet application scaffold from Akhet 1, and what users of that scaffold have later reported doing in their more recent applications.

The demo is distributed separately from Akhet due to its larger number of dependencies and more frequent changes. The Akhet library focuses on stability and backward compatibility, while the demo is free to experiment more and make backward-incompatible changes, and is in a permanent development mode.

### Installation

You can install the demo it from its source repository like any Pyramid application:

```
$ virtualenv --no-site-packages ~/directory/myvenv
$ source ~/directory/myvenv/bin/activate
(myvenv)$ git clone git://github.com/mikeorr/akhet_demo
(myvenv)$ cd akhet_demo
(myvenv)$ pip install -e .
(myvenv)$ pserve development.ini
```

### Features

The demo has the following features which originated in the former 'akhet' scaffold:

- Mako templates.
- Site template to provide a common look and feel to your pages.
- Automatically recognize filenames ending in .html as Mako templates.
- Default stylesheet and browser-neutral reset stylesheet.
- Pylons-like template globals including a helpers module 'h' and a URL generator 'url', and instructions for adding additional ones.
- Serve static files at any URL, without being limited by URL prefixes.
- Listen on localhost:5000 by defalt.
- Beaker session and cache configuration.
- Demonstration of flash messages and logging.

The demo introduces the following new features:

- Class-based views using `@view_config`.
- A pony and a unicorn.

The demo does *not* have these features that were in the former 'akhet' scaffold:

- A SQLAlchemy model. The Pyramid 'alchemy' scaffold and the Models chapter in the Pyramid for Pylons Users guide are sufficient to get started.
- View handlers using 'pyramid_handlers'. Many Akhet users have gone to class-based views using Pyramid's standard `@view_config`, so the demo is doing that now too.
- Subpackages for views and models. These are easy enough to create yourself if you need them.

## 1.2.2 Templates and stylesheets

The demo's templates and stylesheets are designed to function in a variety of environments, so you can copy them to your application as a starting point. The following files are included:

- A home page, *akhet_demo/templates/index.html*
- A site template, *akhet_demo/templates/site.html*
- A stylesheet, *akhet_demo/static/stylesheets/default.css*
- A "reset" stylesheet, *akhet_demo/static/stylesheets/reset.css*

The HTML files are Mako templates. The stylesheets are static files.

### index.html

This is a page template, so it contains only the unique parts of this page. The first three lines are Mako constructs:

```
1  <%inherit file="/site.html" />
2  <%def name="title()">Hello, ${project}!</%def>
3  <%def name="ht_title()">${project}</%def>
```

Line 1 makes the template inherit from the site template, which will add the site's header and footer. Lines 2 and 3 are Mako methods. They output the body title (the <h1> at the top of the page) and the head title (the <title> tag) respectively. Mako templates and methods are not literally Python classes and methods – they compile to modules and functions respectively – but Mako treats them in a way that's similar to classes and methods.

The "${varname}" syntax is a placeholder which will output the named variable. Template variables can come from several sources: (1) keys in the view's return dict, (2) template globals specified in *akhet_demo/subscribers.py*, (3) local variables defined in the template, (4) built-in Mako variables like `self`.

The rest of the file is a big chunk of HTML that will be plugged into the site template. Mako implicitly puts this chunk in a method named "body", which can be called from other templates as we'll see in a moment.

### Site template

The site template contains the "complete" HTML document, with placeholders to plug in content from the page template. The most important placeholder here is "${self.body()}", which outputs the body of the highest-level template in the inheritance chain.

Note the difference between calling "${body()}" and "${self.body()}". The former calls a <%def> method defined in the same template. The latter calls the highest-level <%def> method with that name in the inheritance chain, which may be in a different template.

The site template also calls "self.title()" and "self.ht_title()", and defines default implementations for these methods. The default body title outputs nothing (resulting in an empty title); the default head title is whatever the body title returns. So you can just define a "title" in your pages and forget about "ht_title" if it's the same. But there are times when you'll want to make them different:

- When the body title contains embedded HTML tags like . The head title can't contain these because it will display them literally rather than changing the font.
- Sometimes the body title is too wordy for the head title.
- Many sites want the site's name in the head title. A general rule of thumb is "Short Page Title &emdash; Site Name". Or if you're part of a large organization: "Short Page Title | Site Name | Organization Name". Search engines pay special attention to the head title, so it should contain all the essential words that describe the page, and it should be less than sixty or so characters so it can be displayed in a variety of contexts.

The other kind of placeholder in the site template is "${url.app}", which is used to form static URLs like "${url.app}/stylesheets.default.css". "url" is the URL generator, which the subscriber puts into the template namespace. "url.app" is the application's URL prefix. This is normally empty for a top-level application mounted at "/". But if the application is mounted at a sub-URL like "/site1", that will be what "url.app" is set to.

Normally you'd generate URLs by route name, such as "${url('home')}" or its full form "${url.route('home')}". But static URLs don't have a route name, and the URL generator does not have a `static` method (although you can define one in a subclass). So we're left with literal URLs relative to the application prefix.

The template displays flash messages, which a view may have pushed into the session before redirecting. The code for this is:

```
<div id="content">
<div id="flash-messages">
% for message in request.session.pop_flash():
    <div class="info">${message}</div>
% endfor
</div>
```

The stylesheet displays it all pretty-like.

#### Reset stylesheet

This is an industry-standard reset stylesheet by Eric Meyer, which is in the public domain. The original site is http://meyerweb.com/eric/tools/css/reset/ . It resets all the tag styles to be consistent across browsers.

The top part of the page is Meyer's original stylesheet; the bottom contains some overrides. Meyers does remove some attributes which have generally been assumed to be intrinsic to the tag, such as margins around <p> and <h*>. His reasoning is that you should start with nothing and consciously re-add the styles you want. Some people may find this attitude to be overkill. The reset stylesheet is just provided as a service if you want to use it. In any case, I have re-added some expected styles, and also set <dt> to boldface which is a pet peeve of mine.

If you want something with more bells and whistles, some Pyramid developers recommend HTML5 Boilerplate. It's also based on Meyer's stylesheet.

We're exploring stylesheet compilers like Less, but this version of the demo does not include one.

#### Default stylesheet

This is the stylesheet referenced in the page template; it inherits the reset stylesheet. It defines some styles the default home page needs. You'll probably want to adjust them for your layout.

The bottom section has styles for flash messages. The ".info" stanza is used by the demo. The ".warning" and ".error" styles are not used by the demo but are provided as extras.

### 1.2.3 Details

#### development.ini

The config file contains the following settings which aren't in Pyramid's built-in scaffolds:

- mako.directories: necessary when using Mako, to set the template search path. (Theoretically you don't need this if you specify renderers by asset spec rather than by relative path, but I couldn't get that to work.)
- cache.*: Beaker cache settings. These are not actually necessary because the demo never uses a cache, but they're here for demonstration.
- session.*: Beaker session settings. These are necessary if you use sessions or flash messages.

Beaker supports several kinds of session persistence: in-memory, files, memcached, database, etc. The demo's configuration uses memory mode, which holds the sessions in memory until the application quits. It contains commented settings for file-based sessions, which is Pylons' default. Experienced developers seem to be choosing memcached mode nowadays. Memory sessions disappear when the server is restarted, and work only with multithreaded servers, not multiprocess servers. File-based sessions are persistent, but add the complications of a directory and permissions and maintenance. Memcached avoids all these problems, and it also scales to multiple parallel servers, which can all share a memcached session.

If you copy the session configuration to your application, do change "session.secret" to a random string. This is used to help ensure the integrity of the session, to prevent people from hijacking it.

### Init module and main function

The main function, in addition to the minimal Pyramid configuration, activates Beaker sessions and caching, and sets up templates, subscribers, routes, and a static route. The Beaker setup passes the `settings` dict to Beaker; that's how your settings are read. Pyramid cofigures Mako the same way behind the scenes, passing the settings to it. The "add_renderer" line tells Pyramid to recognize filenames ending in ".html" as Mako templates. The subscribers include we'll see in a minute.

Activating static routes involves an include line and a "config.add_static_route" call.

### Helpers

The demo provides a Pylons-like helpers module, *akhet_demo/lib/helpers.py*. You can put utility functions here for use in your templates. The helper contains imports for WebHelper's HTML tag helpers, but they're commented out. (WebHelpers is a Python package containing generic functions for use in web applications and other applications.) I'm tempted to actually use the tag helpers in the site template but haven't done so yet.

Most of WebHelpers works with Pyramid, including the popular `webhelpers.html` subpackage, `webhelpers.text`, and `webhelpers.number`. You'll have to add a WebHelpers dependency to your application if you want to use it. The only part of WebHelpers that doesn't work with Pyramid is the `webhelpers.pylonslib` subpackage, which depends on Pylons' special globals.

Note that `webhelpers.paginate` requires a slightly different configuration with Pyramid than with Pylons, because `pylons.url` is not available. You'll have to supply a URL generator, perhaps using one of the convenience classes included in WebHelpers 1.3. Paginate's URL generator is *not* Akhet's URL generator: it's a different kind of class specific to the paginator's needs.

### Subscribers

*akhet_demo/subscribers.py* is unique to the demo. It sets up a URL generator and configures several Pylons-like globals for the template namespace. The only thing you need in here is the includeme function, which the application's main function invokes via the `config.include(".subscribers")` line.

The `add_renderer_globals` subscriber configures the following variables for the template namespace:

- `h`: the helpers module.
- `r`: an alias for `request`.
- `url`: the URL generator.

It has commented code to configure "settings", "session", and "c" variables if you want those.

For completeness, here are the system variables Pyramid 1.3 adds to the template namespace:

- `context`: the context.
- `renderer_name`: the name of the renderer.
- `renderer_info`: a `RendererHelper` object (defined in `pyramid.renderers`).
- `request`: the request.
- `view`: the view. (A function or instance.)

As a reminder, everything here is local to the current request. The URL generator is attached to the request object, and the renderer globals are set just before the renderer is invoked. These variables are all discarded at the end of the request.

#### Views

The views module has a base class called `Handler` (but it's not related to "pyramid_handlers"). The index view demonstrates logging, optionally sets a flash message, and invokes a Mako template renderer.

The demo pushes a flash message by calling `self.request.session.flash()` with the message text. By default this puts the message on the "info" queue, and it's displayed using an "info" CSS class. You can push the message onto a different queue by specifying the queue name as a second argument. But that's only useful if the template pops the messages from the other queue by name, otherwise they'll never be displayed. It's customary to name the queues according to the Python logging hierarchy: debug, info (notice), warn(ing), error, critical. The default stylesheet defines CSS classes with distinct styling for several of these levels.

## 1.3 Full changelog

### 1.3.1 2.0 (2012-02-12)

- Move repository to https://github.com/Pylons/akhet and convert to Git format (previously http://bitbucket.com/sluggo/Akhet in Mercurial format).
- Rename all "v" tags, removing the prefix (v1.0.1 -> 1.0.1), so that they sort in Git before the older "pyramid_sqla" tags, and to follow Pyramid's precedent.
- New Akhet demo program distributed separately at https://github.com/mikeorr/akhet_demo . It does not include a SQLAlchemy model, thus completing the break from Akhet's origin in the former "pyramid_sqla".
- Delete 'akhet' application scaffold; the demo replaces it.
- We have a pony. (akhet.pony, based on paste.pony)
- Move non-Akhet-specific parts of the manual to the Pyramid Cookbook, as the "Pyramid for Pylons Users" guide.
- The include enabling static routes is now "akhet.static" instead of "akhet". A backward compatibility shim exists.
- The URL generator's `route` method can generate either an absolute (qualified) URL or a path-only (unqualified) URL, overriding the instance's default mode.

### 1.3.2 1.0.2 (2011-07-20)

- Adjust app skeleton to match URLGenerator.app fix in 1.0.1.

### 1.3.3 1.0.1 (2011-07-18)

- Fix bug in `URLGenerator.app`: it was returning the wrong value and was documented wrong.

### 1.3.4 1.0 (2011-04-04)

- App skeleton:
  - Simplify home page and add a Mako site template that can be easily extended by the user. New documentation chapters.
  - New default layout and stylesheet by Marcin Lulek (Ergo^), designed to be extensible and a learning tool.
  - Separate industry-standard "reset" stylesheet for cross-browser consistency.
  - Add "flash message" demo to home page.
  - Add "requirements" file for easy installation of dependencies.

### 1.3.5 1.0b2 (2011-03-19)

- App skeleton:
  - Add Beaker cache configuration
  - Fix bug in urlgenerator: missing variable assignment

### 1.3.6 1.0b1 (2011-03-19)

- Rename distribution to Akhet and app template to akhet.
- Delete all code pertaining to the SQLAlchemy library, which is now in the "SQLAHelper" package.
- `URLGenerator` makes generating route URLs and other application URLs more convenient.
- App template:
  - Change `handlers` to a package and refactor for larger applications.
  - Change `models` to a package.
  - Create a `lib` package and move helpers.py to it as Pylons does.
  - Add commented examples of advanced usages in init and base handler.
  - The `url` template global is now a URLGenerator instance. You can still call it as before to generate a route URL but don't pass the `request` arg any more. The URL generator is also available in views as `self.request.url_generator`.
  - Create the SQLAlchemy engine ourself; SQLAHelper no longer does this.
  - Change "[app:{{project}}]" to "[app:myapp]" in INI files so that the name is well known and easier to type on the command line (e.g., for 'pshell').
  - Ask whether to configure SQLAlchemy.
  - Switch to `pyramid_tm` transaction manager from `repoze.tm2`.
- 'akhet/tests/make_test_app.sh' is a quick-and-dirty script to create and run a test application.

### 1.3.7 Repository Akhet created

Repository "Akhet" was cloned from "pyramid_sqla" at this point. All tags "vVERSION" were renamed to "pyramid_sqla-VERSION". A new tag "pyramid_sqla-dev" points to the last code change before the split.

### 1.3.8 pyramid_sqla-dev (never released; changeset c0c74051c201)

- `add_static_route` is now a Pyramid config method if you call the new `includeme` function. This is used in the application template.
- Add `pyramid_sqla` as a dependency in the application template.
- Delete websetup.py. Console scripts are more flexible than "paster setup-app".
- Fix but that may have prevented create_db.py from finding the INI stanza.
- 100% test coverage contributed by Chris McDonough.
- Delete unneeded development code in static.py.
- Set Mako's 'strict_undefined' option in the application template.

### 1.3.9 pyramid_sqla-1.0rc1 (2010-01-26)

- 'pyramid_sqla' application template supports commit veto feature in repoze.tm2 1.0b1.
- Add production.ini to application template.
- Delete stray files in application template that were accidentally included.

### 1.3.10 pyramid_sqla-v0.2 (2011-01-19)

- Pyramid 1.0a10 spins off view handler support to 'pyramid_handlers' package.
- 'pyramid_sqla' application template depends on Pyramid>=1.0a10.

### 1.3.11 pyramid_sqla-0.1 (2011-01-12)

- Initial release.
- Warning: a change in Pyramid 1.0a10 broke applications created using the this version's application template. To run existing applications under Pyramid 1.0a10 and later, add a 'pyramid_handlers' dependency to the `requires` list in setup.py and reinstall the application.

## 1.4 Appendix: Rant about scaffolds and PasteScript

The main reason the 'akhet' scaffold is gone is that maintaining it turned out to be a significant burden. Testing a scaffold requires several manual steps – change a line of code, generate an app, install it, test a URL, test some other URLs, change the application, backport the change to the scaffold, generate another app, install and test it, -OR- make changes directly to the scaffold and generate an app to see whether it works. If it requires custom application code to trigger the bug, you have to re-apply the code every time you crete the app. Beyond that, Pyramid evolves over time, so the scaffolds have to be updated even if they were working OK. And the scaffold API is primitive and limited; e.g., you can't inherit from a scaffold and specify just the changes between yours and the parent.

The final barrier was Python 3. Other packages descended from Paste have been ported to 3 (PasteDeploy, WebOb), but Paste and PasteScript haven't been. There doesn't seem to be much point because the scaffold API needs to be overhauled anyway, many of paster's subcommands are obsolete, and some people question the whole concept of plugin subcommands: what exactly is its benefit over bin scripts?

Pyramid 1.3 drops the Paste and PasteScript dependencies, and adds bin scripts for the essential utilities Pyramid needs: 'pcreate', 'pserve', 'pshell', 'proutes', 'ptweens', and 'pviews'. These were derived from the Paste code, and the scaffold API is unchanged.

Two other factors led to the demise of the scaffold. One, users wanted to mix and match Akhet features and non-Akhet features, and add databases to the scaffold (e.g., MongoDB). That would lead to more questions in the scaffold, or more scaffolds, and more testing burden (especially since I didn't use those databases).

The other factor is, I began to doubt whether certain Akhet features are necessarily better than their non-Akhet conterparts. For instance, Akhet 1 and Pyramid have different ways of handling static files. Each way has its pluses and minuses. Akhet's role is to make the Pylons way available, not to recommend it beyond what it deserves.

So faced with the burden of maintaining the scaffold and keeping it updated, I was about to retire Akhet completely, until I realized it could have a new life without the scaffold. And as I work on my own applications and come up with new pieces of advice or new convenience classes, I need a place to put them, and Akhet 2 is an ideal place. So viva the new, scaffold-free, Akeht 2.