# Kay Framework

## What's this

Kay is a web framework completely specialising on Google App Engine. But basic design is based on the Django framework, like middleware, settings and pluggable application, etc. Kay uses Werkzeug as lower level framework, uses Jinja2 as template engine, uses babel for handling language translations. This software is distributed under BSD license. See LICENSE for the details.

## The software that Kay bundles

- babel
- jinja2
- simplejson
- werkzeug
- pytz

## Dependencies

- Google App Engine SDK

  If you installed zip version of the SDK(most unix style operation system users are so), don't forget to create a symlink pointed to the real SDK at /usr/local/google_appengine.

```
$ sudo ln -s /some/whare/google_appengine /usr/local/google_appengine
```

## Quick Start

- First, make symlynk of kay directory inside your project directry, copy settings files.

```
$ hg clone http://bitbucket.org/tmatsuo/kay/
$ mkdir yourproject
$ cd yourproject
$ ln -s ../kay/kay .
$ ln -s ../kay/manage.py .
$ cp ../kay/urls.py .
$ cp ../kay/settings.py .
$ cp ../kay/app.yaml .
```

- Second, you can create your first application with following command.

```
$ python manage.py startapp hello
$ vi urls.py
$ vi settings.py
```

There is a sample urls.py in the file hello/urls.py as a comment. You have to add 'hello' to the INSTALLED_APPS tupple in the settings.py.

- urls.py

```
from hello import urls as hello_urls

def make_url():
  return Map([
    Submount('/hello', hello_urls.make_rules())
  ])

all_views = {
}

all_views.update(hello_urls.all_views)
```

- settings.py

```
INSTALLED_APPS = (
  'kay.sessions',
  'hello'
)
```

- Run your application

```
$ python manage.py runserver
```

- Upload your application

```
$ python manage.py appcfg update
```

- You can handle i18n like following.

```
$ python manage.py extract_messages hello
$ python manage.py add_translation hello -l ja
$ vi hello/i18n/ja/LC_MESSAGES/messages.po
$ python manage.py compile_translations hello
```

You can also merge newly added catalogue into your translations as follows.

```
$ python manage.py extract_messages hello
$ python manage.py update_translation hello -l ja
$ vi hello/i18n/ja/LC_MESSAGES/messages.po
$ python manage.py compile_translations hello
```

# Shell tools

- Invoking *python manage.py shell* gives you python (or ipython if available) console session with the same DatastoreFileStub settings of local dev server.

**Note:**

The local dev server reads datastore data file only on startup. So, the dev server will never notice about the datastore operation on your console session. You must restart your dev server for reflecting the result of the console sessions.

- Invoking *python manage.py rshell* is the same as above except for using RemoteDatastore stub. You can access the data on the production server.

**Note:**

Be careful when use this feature.

# Datastore

- You must use GAE models directly. You can use kay.utils.forms for form handling. You can construct a form automatically from the model definition with kay.utils.forms.modelform.ModelForm.

  Unfortunately, for now, there is no documentation about forms package yet. Please see kay.tests.formtest for examples. Sorry for the inconvinience.
- By default, db.Model.kind() returns ('model's app name' + _ + 'model name').lower(). So when you see the management console, there will be 'appname_modelname' style kind names . Please don't be surprised with those names.

  You can change this behaviour by settings ADD_APP_PREFIX_TO_KIND to False in your settings.py.

# Forms

- To define form class, you can define a class that extends kay.utils.forms.Form. For example the code bellow will give you the form contains two text fields with different validators.

```
from kay.utils.forms import Form
class PersonForm(Form):
  name = TextField(required=True)
  age = IntegerField()
```

You can use this form in your view like following.

```
from forms import PersonForm
form = PersonForm()
if request.method == 'POST'
  if form.validate(request.form, request.files):
    name = form['name']
    age = form['age']
    do something with valid form ...
  else:
    do something with invalid form ...
```

- You can also use ModelForm to create a form automatically from Model class.

```
from google.appengine.ext import db

class MyModel(db.Model):
  name = db.StringProperty(required=True)
  age = db.IntegerProperty()

from kay.utils.forms.modelform import ModelForm

class MyForm(ModelForm):
  class Meta:
    model = MyModel
```

- For more details about forms, please refer to the docs/forms-usage.rst.

# Questions and Bug Report

- Please visit Kay framework google group. http://groups.google.com/group/kay-users
- Or, contact the project leader directly. Takashi Matsuo <tmatsuo@candit.jp>
- Code site http://code.google.com/p/kay-framework/