

PimenTech libcommonDjango API

Authors : Guillaume Savary, Julien Herbin, Arnaud Rouanet, Stanislas Guerra, Frederic de Zorzi
Contact : info@_nospam_pimentech.net
Revision : 8419
Date : 2010-02-26
Copyright : This document has been placed in the public domain.
Tags : django common api pimentech english calendar

Installation

This package require *python-docutils*

```
svn checkout http://svn.pimentech.org/pimentech/libcommonDjango
cd libcommonDjango
make install
```

Trac access : <http://trac.pimentech.org/pimentech/browser/libcommonDjango>

Django admin modules

CSV and Stats modules

We have started to add some modules to admin interface. The *CSV module* allow to import export data files per table, in admin fields order. It supports multiple encodings.



FIG. 1 – New buttons on admin lists.

Add/Change stages

The admin add/change stage function have been overloaded by pim change/add stage functions : if *creation_auth_user* and *modification_auth_user* fields are present in table, these fields are updated with logged user.id.

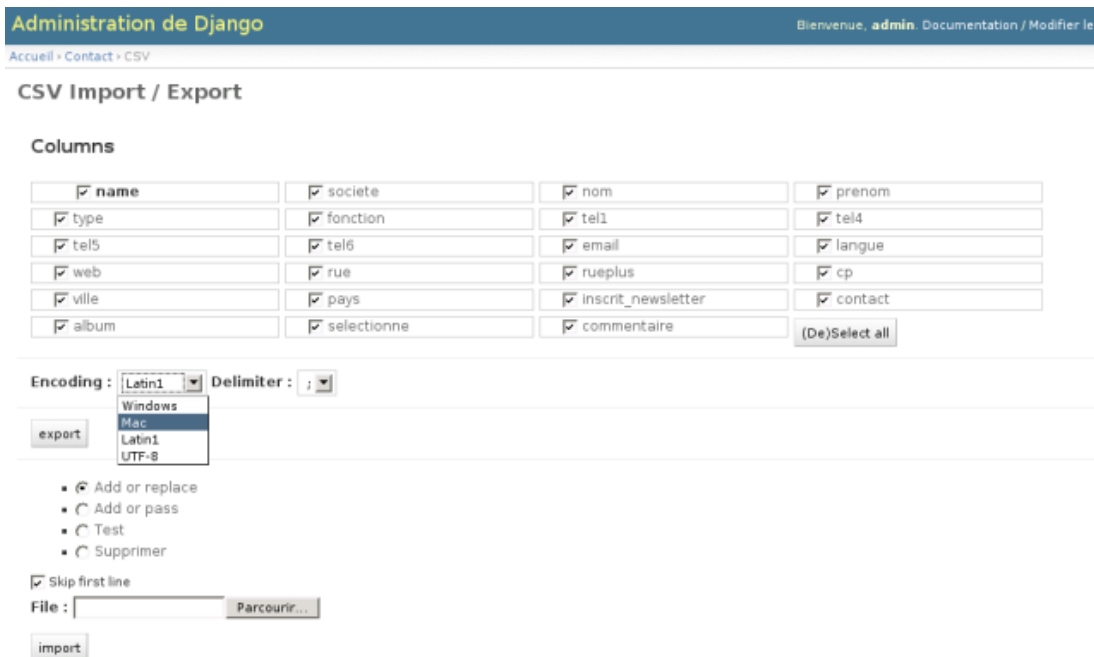


FIG. 2 – CSV Import/Export form

Installation

- in settings.py, put 'django.pimentech.common' in INSTALLED_APPS list, before 'django.contrib.admin'.
- in urls.py, replace 'django.contrib.admin.urls' by 'django.pimentech.common.urls'

Custom Fields

ReStructured text field

DateField

See http://svn.pimentech.org/pimentech/libcommonDjango/django_pimentech/fields.py

FrenchDateField

This field allows you to display the date in the french format (dd/mm/yyyy). It also provides extra options to purpose a calendar beside the field.

In order to use the calendar, you have to include the following javascripts in your header (adapt the urls according to your own settings) :

index.html

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Django_calendar Demo</title>
    <script type="text/javascript" src="/admin/js118n/"></script>
    <script type="text/javascript" src="/admin_media/js/core.js"></script>
    <script type="text/javascript" src="/admin_media/js/admin/RelatedObjectListTools.js"></script>
    <script type="text/javascript" src="/admin_media/js/calendar.js"></script>
    <script type="text/javascript" src="/pimentech/js/django_calendar.js"></script>
```

```

        <!-- Here a style for the calendar-->
        <style type="text/css">
            .calendarlink { padding-left:12px; background:url(/static/img/icon_ca
        </style>

    </head>
</html>

```

The `django_calendar.js` scripts is provided by the [Pimentech Scripts library](#)
 Here an example of a view using the `FrenchDateField` :

views.py

```

from django.pimentech.fields import *

class MyForm(Form):
    """A basic form."""
    ...
    my_french_date = FrenchDateField(required=True, initial=date.today(),
        widget=FrenchDateInput(attrs={'class':'type-date', 'calendar':True, \
        'calendar_c

def my_view(request):
    ...
    form = MyForm()
    return render_to_response('my_app/my_template.html',
        {'form':form,},
        context_instance=RequestC

```

The template file may looks like that :

my_template.html

```

<?xml version="1.0" encoding="utf-8"?>

<form id="my_form" name="my_form" action="{{ request.path }}" method="post">
    {{ form.my_french_date }} {{ form.errors.my_french_date }}
    <input type="submit" name="submit" value="Go!" />
</form>

```

Et voilà :

Utils

Function : `get_object_or_404(klass, args, kwargs)`

Replaces django's `get_object_or_404()` function (too much SQL joins).

Example :

```

from django.pimentech.utils import get_object_or_404
K = get_object_or_404(Klass, uid=1)

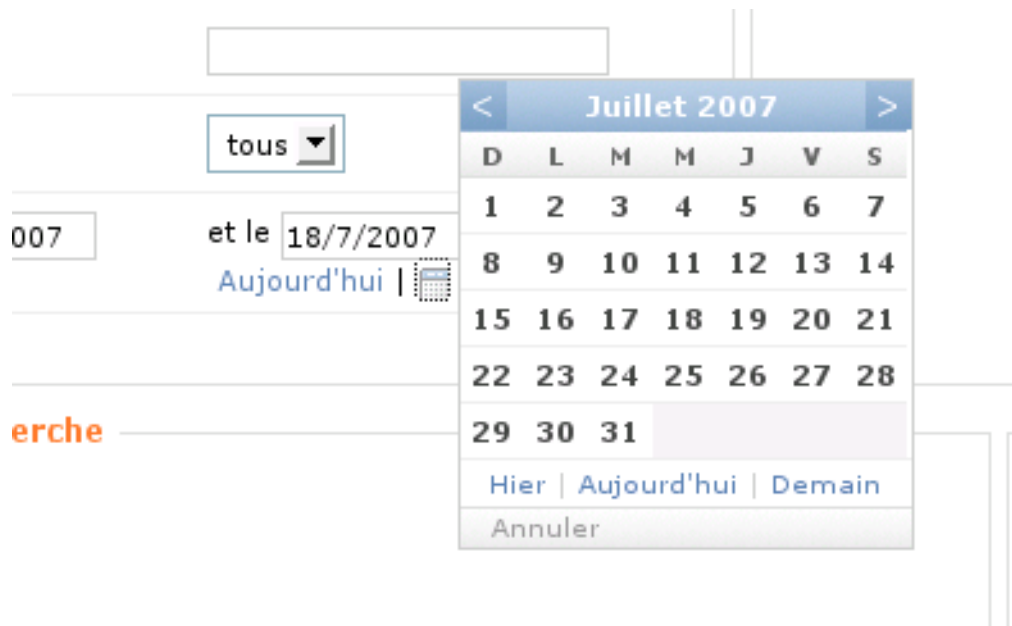
```

Function : `send_form(form, sender, recipients, subject)`

Send newform content by e-mail.

In progress

Sorry for the lack of documentation, we are too busy for the moment..



Backoffice form modules with validating functions

http://svn.pimentech.org/pimentech/libcommonDjango/django_pimentech/validationforms.py

With this module, you can do modifications on your database with post-validation : if a row is modified, a copy is generated for post (un)validation. Works with 1-n and n-n relations.

DbHandler, a Django-ORM alternative

http://svn.pimentech.org/pimentech/libcommonDjango/django_pimentech/dbhandler.py

If like us, you create django applications on existing models, using the Django ORM is often inadequate : database model is not application oriented. With this module, you can build application-oriented classes with sql queries.

Log

Loguer des messages dans le code

Fonctions permettant d'écrire des messages sur stdout en mode 'runserver' ou dans le fichier de log 'error' d'Apache. Les fonctions sont :

- message() - logue avec le flag APLOG_NOTICE
- warning() - logue avec le flag APLOG_WARNING
- error() - logue avec le flag APLOG_ERR

Exemple :

```
import django.pimentech.log as log
l = (1, 2, 3)
log.message(l)
```

Loguer les erreurs Django

Le middleware ErrorLogMiddleware permet de stocker les erreurs Django dans /var/log/django. Exemple de fichier settings.py :

```
MIDDLEWARE_CLASSES = (
    'django.middleware.common.CommonMiddleware',
    'django.pimentech.errorlog_middleware.ErrorLogMiddleware',
)
```

Ensuite un simple lien dans /var/www vers /var/log/django permettra de récupérer les erreurs au format HTML.

Network

Classe : JSONRPCService

Cette classe permet de créer un service JSON-RPC en Django (<http://json-rpc.org/>). Le code est largement inspiré de celui trouvé sur le wiki de Pyjamas (<http://trac.pyworks.org/pyjamas/wiki/DjangoWithPyJamas>) à la différence que le paramètre `request` est transmis aux vues distantes, comme pour les vues classiques.

L'exemple suivant montre comment créer une vue invoquable par JSONRPC. Celle-ci prend une chaîne de caractère en paramètre et renvoie la même chaîne en majuscules.

Fichier `stringservices/views.py` :

```
from django.pimentech.network import *

service = JSONRPCService()

def to_upper(request, str):
    return str.upper()
service.add_method("to_upper", to_upper)
```

Note

A partir de python 2.4, il est possible d'utiliser le décorateur `jsonremote` pour un code plus élégant.

Fichier `stringservices/views.py` (python >= 2.4) :

```
from django.pimentech.network import *

service = JSONRPCService()

@jsonremote(service, 'to_upper')
def to_upper(request, str):
    return str.upper()
```

Dans le fichier `urls.py`, il suffit d'ajouter une règle pour transmettre les requêtes JSON-RPC à notre objet `JSONRPCService` :

```
(r'^/chat-service-jsonrpc/$', 'stringservice.views.service'),
```

logprofile

Ce module permet de faire du profiling. Il nécessite le module non libre `python-profiler`.

- `create_profile(name)` : renvoie un objet de profiling
- `log_stats(name, *sort, *result)` : tableau des stats
- `log_callers(name, *sort, *result)` : stats par fonction appelée

Exemple :

```
from django.pimentech.logprofile import *

prof = create_profile("/tmp/file.prof")
# on peut ainsi recuperer le fichier de profile pour analyse
```

```

prof.start()

... some python code ...

prof.stop()
prof.close()

# standard (time, call)
profile_stats = log_stats("/tmp/file.prof") + "\n"

# filtre sur une fonction et 25% des resultats :
profile_stats += log_stats("/tmp/file.prof", result=['funcname', 0.25]) + "\n"

# filtre avec tri par nom de fonction puis par nombre d'appels:
profile_stats += log_stats("/tmp/file.prof", sort=['name', 'calls'] + "\n"

print profile_stats

```

Voir <http://www.python.org/doc/2.3/lib/profile-stats.html> pour plus d'infos sur les tris et résultats possibles.

Voir également <http://code.djangoproject.com/wiki/ProfilingDjango> pour la page officielle.

Templatetags

Un ensemble de templatetags réutilisables sont fournis par l'application `gdjango.pimentech.common` (à ajouter dans la variable `INSTALLED_APPS` de `settings.py`).

Navigation page à page

Pour utiliser le module de navigation par pages, il suffit de mettre dans le template :

```

{% load pimentech_extras %}

{% pages_navigation nb_results page nb_per_page %}

```

Les trois paramètres à passer à `pages_navigation` sont : le nombre total de résultats, le numéro de la page courante, et le nombre de résultats par page (15 si non spécifié).

Les variables suivantes sont fournies au template chargé de représenter le module de navigation :

- `page` : le numéro de la page courante
- `prev_pages` : la liste des 4 pages précédant la page courante (ou moins si `page < 4`).
- `prev_page` : la page précédente, ou `None` si pas de précédent.
- `next_pages` : la liste des 4 pages suivant la page courante (ou moins)
- `next_page` : la page suivante, ou `None` si pas de suivant.

Un template par défaut est fourni dans **libcommonDjango**. Il est possible de le modifier en redéfinissant localement le template `common/pages_navigation.html`.

Voici le template par défaut :

```

{% load i18n %}

{% if prev_pages %}
<a href="?{% append_get %}page={{ prev_page }}" title="{% trans 'Previous Page' %}
{% for p in prev_pages %}
<a href="?{% append_get %}page={{ p }}" title="{% trans 'Page' %} {{ p }}">{{ p }}
{% endfor %}
{% else %}
<lt;&lt;
{% endif %}

```

```

{% trans 'Page' %} {{ page }}
{% if next_pages %}
{% for p in next_pages %}
- <a href="?{{ append_get }}page={{ p }}" title="{% trans 'Page' %} {{ p }}">{{ p }}
{% endfor %}
<a href="?{{ append_get }}page={{ next_page }}" title="{% trans 'Next Page' %}">&
{% else %}
&gt;&gt;
{% endif %}

```

Cache

Fonction : `cache_force_view_expiration(path, key_prefix=None)`

Cette fonction permet de forcer l'expiration d'une vue dans le cache Django. Elle prend en paramètre une URL absolue, menant à la vue que l'on souhaite supprimer du cache. Elle prend également un paramètre optionnel `key_prefix`, qui a comme valeur par défaut `settings.CACHE_MIDDLEWARE_KEY_PREFIX` (il permet de générer la signature de la vue dans le cache). Si une vue peut être invoquée depuis deux chemins différents avec les mêmes paramètres, seule la page dont le chemin a été passé à `cache_force_view_expiration` sera recalculée. `cache_force_view_expiration` peut par exemple être invoquée suite à la modification d'un objet afin qu'une page soit recalculée au prochain hit.

Fichier `views.py` :

```

from django.views.decorators.cache import cache_page

def ma_vue_cachee(request, param):
    (...)
    return (...)

ma_vue_cachee = cache_page(ma_vue_cachee, 60 * 15) # expiration de la version cach

```

Fichier `urls.py` :

```

from django.conf.urls.defaults import *

urlpatterns = patterns('mon_appli.views',
    (r'^ma-vue-cachee-(?P<numero>\d+)/$', 'ma_vue_cachee'),
    (...))

```

Demande d'expiration explicite d'une vue :

```

from django.pimentech.cache import cache_force_view_expiration

cache_force_view_expiration('/mon-appli/ma-vue-cachee-18/')

```